Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА (ПРОЕКТ)

на тему:

Розробка системи обліку та аналізу успішності студентів КНТЕУ засобами блокчейн

Студента 2 курсу, 5м групи,
спеціальності 121 "Інженерія програмного забезпечення"
спеціалізації "Інженерія програмного забезпечення" _____ Кубишева Дмитра Вячеславовича
підпис студента

Науковий керівник
науковий ступінь, вчене звання _____ Цюцюра М.І., к.т.н, доцент
підпис керівника

Гарант освітньої програми
науковий ступінь, вчене звання _____ Криворучко О.В., д.т.н., професор
підпис керівника

КИЇВ – 2019

ЗМІСТ

					КНТЕУ 121– 05-15.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*	*Розробка системи обліку та аналізу успішності студентів КНТЕУ засобами блокчейн*	*Стадія*	*Аркуш*	*Аркушів*
Зав. кафедри		Криворучко О.В.				Зміст	*2*	*56*
Керівник		Цюцюра М.І.						
Гарант		Криворучко О.В.			*Зміст*	Факультет обліку, аудиту та інформаційних систем, 2м курс, 5 група		
Розроб.		Кубишев Д.В.						

ВСТУП

Сьогодні відбувається потужний розвиток інформаційних систем та технологій і як наслідок їх широке застосування в різних сферах діяльності суспільства. Значна кількість сучасних державних та приватних установ використовує інформаційні системи для управління виробничими процесами, підтримки прийняття рішень, збереження та обробки інформації, пошуку необхідних даних тощо. Майже всі ці системи працюють за принципом, коли управління процесами відбувається централізовано й повний контроль над системою можна здобути отримавши доступ до головного центрального сервера. Внаслідок чого збільшується ризик компроментації всієї системи, кількість уразливостей та загроз ІТС.

Кожен напевно чув термін «блокчейн». Блокчейн - це розподілена база даних, яка не підключена до загального сервера. Ця база даних містить в собі перелік впорядкованих записів, кількість яких постійно збільшується. Ці записи називаються «блоки». Кожен такий блок містить мітку часу, а також посилання на попередній блок. Технологія блокчейна передбачає використання унікального типу шифрування, яке дозволяє захищати певні частини ланцюжків блоків. Таким чином, внесення в них будь-яких змін стає неможливим. Технологія блокчейна набирає все більшої популярності в наші дні: все більша кількість підприємств і організацій розуміють цінність ланцюжка блоків і можливостей їх використання. У випускно-кваліфікаційній роботі будуть розглянуті можливості технології для освітнього процесу.

					КНТЕУ 121– 05-15.МР		
					Розробка системи обліку та аналізу успішності студентів КНТЕУ засобами блокчейн	*Стадія*	*Аркуш*	*Аркушів*
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*		Зміст	*3*	*56*
Зав. кафедри		Криворучко О.В.						
Керівник		Цюцюра М.І.						
Гарант		Криворучко О.В.			*Зміст*	Факультет обліку, аудиту та інформаційних систем, 2м курс, 5 група		
Розроб.		Кубишев Д.В.						

Актуальність роботи: полягає у створенні децентралізованого веб-сервісу обліку та аналізу успішності студентів.

Об'єкт дослідження: освітній процес КНТЕУ.

Предмет дослідження: методи та алгоритми створення програмного забезпечення обліку та аналізу успішності студентів КНТЕУ засобами блокчейн.

Мета роботи: розробка та впровадження системи обліку та аналізу успішності студентів КНТЕУ засобами блокчейн.

Методи і технології дослідження: технологія блокчейн, фреймворк Angular.JS.

Завдання дослідження:

- аналітичний огляд предметної області;
- написання технічного завдання;
- розгляд створення блокчейн веб-сервісу Angular;
- розробка use-case для веб-сервісу;
- розробка веб-сервісу;
- тестування створеного веб-сервісу.

Наукова новизна дослідження: розробка децентралізованної інформаційної системи для потреб освітнього процесу.

Результати роботи: створений веб-сервіс обліку та аналізу успішності студентів.

					КНТЕУ 121– 05-15.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*	*Розробка системи обліку та аналізу успішності студентів КНТЕУ засобами блокчейн*	*Стадія*	*Аркуш*	*Аркушів*
Зав. кафедри		Криворучко О.В.				Зміст	*4*	*56*
Керівник		Цюцюра М.І.						
Гарант		Криворучко О.В.			*Зміст*	Факультет обліку, аудиту та інформаційних систем, 2м курс, 5 група		
Розроб.		Кубишев Д.В.						

РОЗДІЛ І.
АНАЛІЗ ТА ОБГРУНТУВАННЯ ВИБОРУ ВИКОРИСТОВУВАНИХ ТЕХНОЛОГІЙ

1.1 Загальні принципи та переваги технології блокчейн

Сучасний етап розвитку світового суспільства щільно пов'язаний зі стрімким розвитком інформаційних технологій. Складність інформаційних систем (ІС) невпинно зростає, відповідно набувають актуальності питання щодо ефективного управління процесами створення, тестування та впровадження таких систем. Комплексне вирішення цієї проблема – складна і довготривала робота, що потребує високої кваліфікації задіяних у ній працівників. Адже, незважаючи на великий прогрес, багато питань у сфері автоматизації проектування ІС не піддається повній автоматизації і виконується на інтуїтивному рівні, на досвіді та прогностичних здібностях фахівців, експертних оцінках та експериментальній перевірці якості функціонування системи.

Слід зауважити, що експлуатаційні витрати на впровадження й супроводження складної ІС можуть перевищувати витрати на її розробку. Досвід свідчить, що пошук помилок, які відбувалися на ранніх стадіях аналізу та проектування системи з часом зростає у геометричній прогресії. Наприклад, помилки передпроектувального аналізу, що виявлені на стадії реального проектування, «коштують» приблизно вдвічі дорожче. Виявлення таких помилок на етапі тестування – вже у 10 разів дорожче. А усунення

помилок на етапі експлуатації може коштувати у 100 разів дорожче, ніж своєчасне виявлення їх на етапі аналізу системи. Крім того, знаходження системних помилок на етапі експлуатації пов'язано з роботою замовників (тобто реальних користувачів), що негативно впливає на імідж проекту та подальшу експлуатацію системи. Тому вибір методології проектування, засобів супроводження інформаційної системи, тестування її окремих складових на всіх стадіях розробки є складним та відповідальним питанням, від якого залежить успішність та ефективність проекту загалом.

В останні роки представники державних структур, бізнесу, фінансової сфери, банківської спільноти виявляють великий інтерес до відносно нової технології - блокчейн. Спочатку ця технологія згадувалася виключно в зв'язку з таким феноменом, як криптовалюта, перша з яких - Bitcoin - з'явилася у 2009 році. У 2013 р в центрі уваги опинилися додатки «Blockchain 2.0», що стосуються найрізноманітніших сфер - від реєстрації доменного імені до фінансових контрактів, краудфандінга і навіть ігор, засновані на тих же технологіях, що забезпечують децентралізацію та безпеку. За визначенням Д. і А. Тепскоттів, авторів книги «Революція блокчейна», «... блокчейн - це вічний цифровий розподілений журнал економічних транзакцій, який може бути запрограмований для запису не тільки фінансових операцій, але і практично усього, що має цінність». Іншими словами, блокчейн (або ланцюжок блоків) являє собою розподілену базу даних, у якій пристрої зберігання даних не підключені до загального сервера. Така база даних зберігає безперервно зростаючий список упорядкованих записів, так званих блоків, кожному з яких присвоєна мітка часу і посилання на попередній блок. Блокчейн - це мета-технологія, тому що вона впливає на інші технології і складається з декількох технологій. Архітектурні шари блокчейна: база даних, програмне додаток, кілька комп'ютерів, підключених

один до одного, клієнти, що мають доступ до нього, програмне середовище, на якій він заснований, інструменти для контролю над ним.

Чим технологія блокчейн привертає державні і бізнес-структури? По-перше, безпека. Блокчейн - це захищений цифровий реєстр, мережа рівних вузлів, у якій зберігаються транзакції з передачі прав власності на об'єкти, а не бази даних об'єктів власності (наприклад, рахунки клієнтів з розміщеними на них засобами). Функції перевірки і зберігання транзакцій розподіляються між безліччю вузлів без участі регуляторів. Транзакції об'єднуються в ланцюжки, порядок яких фіксується в ланцюжках блоків (звідси і назва - Blockchain). У кожному блоці містяться відомості про хєш-код попереднього блоку, що захищає весь ланцюжок від змін (рис 1.1).

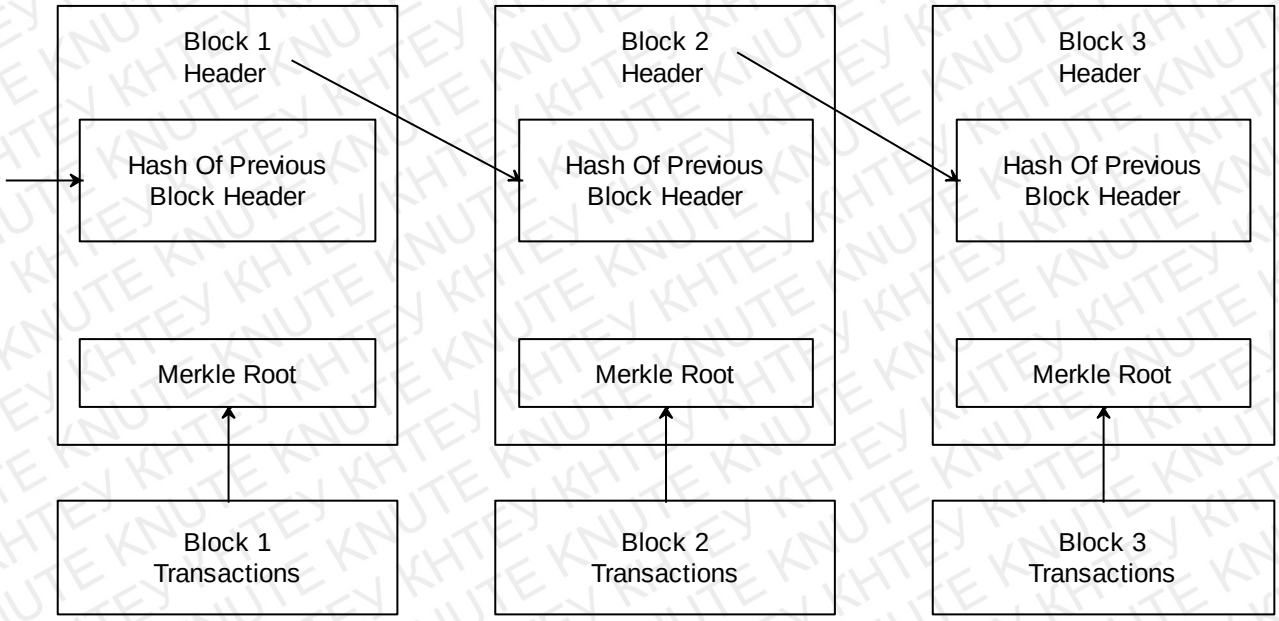

Рис.1.1 Вміст блоку та ланцюг блоків.

Велике число вузлів і криптографічний алгоритм роблять підміну інформації практично неможливою. Ця особливість усуває такі недоліки більшості існуючих інфраструктур, як централізованість, обов'язкова

наявність довірчих відносин між усіма гравцями ринку, участь регуляторів для обміну інформацією.

По-друге, впровадження інфраструктури на основі блокчейну дозволило б істотно знизити витрати на її підтримку і нівелювати численні ризики, пов'язані з безпекою. Відсутність посередників дозволяє заощадити кошти всім взаємодіючим сторонам. Це підтверджується результатами аналізу операційних витрат 50 банківських організацій, проведеного компаніями McLagan і Wirex: - за рахунок оптимізації якості даних, підвищення рівня прозорості та внутрішнього регулювання вдалося скоротити витрати на фінансову звітність на 70%; - спрощення звірки фінансових транзакцій дозволяє скоротити витрати на 30-50%; - спрощення спільного доступу до клієнтських даних призводить до скорочення витрат на централізовану діяльність в два рази; - результатом зниження потреби в звірці угод і пошуку помилок, часткової автоматизації діяльності фахівців по взаєморозрахунках і клірингу є скорочення на 50% витрат на бізнес-операції.

По-третє, блокчейн дозволяє замінити численні моделі узгодження даних і таким чином суттєво прискорити будь-які процеси. Наочним прикладом є проведення міжнародного акредитива між S7 Airlines і «Альфа Банком» у вигляді транзакції через блокчейн Ethereum за 23 секунди замість звичайних 14 днів.

По-четверте, важливою перевагою блокчейна є універсальність. З його допомогою можна створювати громадські бази даних: земельні реєстри, відкриті ресурси для реєстрації прав власності, в тому числі інтелектуальної, управління енергетичними потоками, голосування через інтернет. Все більше поширюються розумні контракти (Smart Contracts) - транзакції, які автоматично виконуються при настанні запрограмованого спочатку набору умов (рис 1.2).

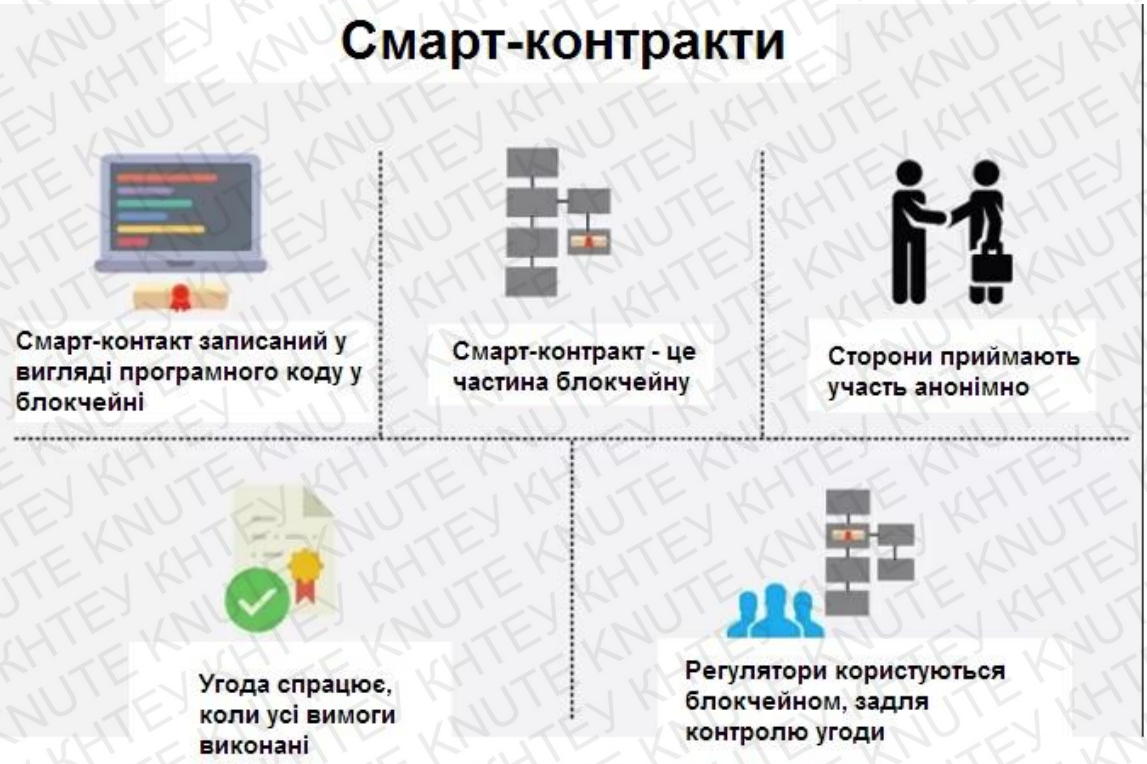

Рис 1.2 Принципи смарт-контрактів

Блокчейн, як і будь-яка технологія, не досконалий, у нього є деякі явні недоліки, особливо в плані масового впровадження технології. Так, варто назвати його невизначений нормативний статус. Блокчейн і криптовалюта знаходяться за межами законодавчого регулювання більшості країн та з економічної точки зору блокчейн наділений певними недоліками:

- Висока енергозалежність найпоширенішого блокчейна з алгоритмом консенсусу Proof-of-Work за рахунок складності транзакції, що робить його дорогою технологією;
- При передачі електронних цінностей блокчейн дозволяє істотно заощадити на оплаті послуг посередників і гарантів. Однак саме створення системи та впровадження її в будь-яку сферу є дуже витратним;

- Масштабованість є ще одним обмеженням через розмір публічного блокчейну. У разі перевантаженості бази швидкість переказів істотно знижується. Наприклад, в системі Bitcoin одна транзакція може відбуватися 4-5 годин і більше;
- Диференціація блокчейна. В даний час існує більше 3000 цифрових монет, багато з яких мають свої власні версії блокчейна. Поки неясно, які саме з них зможуть вижити і розвинутися в майбутньому, які з них віддадуть перевагу використовувати розробники і великі компанії, а які підуть у небуття.

Серед технічних недоліків можна виділити наступне:

- Вплив на рівні мережі: - хакерська атака DDoS (Distributed Denial of Service), або розподілена атака типу «відмова в обслуговуванні»; - «атака Сивіли»; - Eclipse attack, або «атака інформаційного затемнення»;
- Вплив на рівні користувача: ботнети, які поширюються через дропери - спеціальні анонімні шкідливі програми, які маскуються під піратські версії лінцензіонних програм. З юридичної точки зору вразливість на рівні користувача пов'язана з деанонімізацією учасників ринку;
- Вплив на цілісність блокчейну: - «Атака 51%»; - Double spending, або подвійна витрата, яка має на увазі двічі успішне використання одних і тих же коштів; - Selfish mining, або селфіш-Майнінг - стратегія видобутку Bitcoin, коли користувачі мережі за окремою угодою об'єднуються в групи з метою збільшення власного доходу. Дана дія може централізувати мережу і вбити початкову концепцію децентралізованої системи;

- Атаки, що не залежать від блокчейну і застосовні до всіх мережних технологій: - фішинг. За даними Group-IB, в 2017 р більш 50% коштів з блокчейн-проектів вкрали за допомогою фішингу, в квітні 2018 р хакери викрали 150 млн дол. з адрес кріптовалютного гаманця MyEtherWallet; - дефейс - злом сайтів блокчейн-проектів і підміна адреси для збору коштів на заслання своїх гаманців. У липні 2017 р ізраїльський стартап CoinDash втратив близько 40 000 монет Ethereum (понад 7 млн дол. по курсу на період злому). Технічно блокчейн - це база даних, яка представляє собою розподілений реєстр з можливістю відкритої перевірки.

З точки зору бізнесу блокчейн - це обмінна мережа для переміщення транзакцій, вартості, активів між рівними партнерами, без допомоги посередників. З юридичної точки зору, блокчейн перевіряє транзакції, замінюючи (а точніше, роблячи непотрібними) контролюючі органи. Експерти вважають, що впровадження цієї технології щодо можливого ефекту може не поступатися відкриттю Інтернету на початку XXI сторіччя. Блокчейн дозволить без посередників організувати торгівлю, впровадити безліч сервісів в повсякденне життя, змінити роботу банківської сфери. Технологія блокчейн як мінімум збільшує ефективність того, що вже відбувається. Незважаючи на велику кількість успішних прикладів, ще не всі можливості і особливості блокчейну вивчені до кінця. Його впровадження найчастіше обмежується стадіями тестування і заявами про застосування. Втім, експерти впевнені в ефективності блокчейна і пророкують технології велике майбутнє.

1.2 Використання технології блокчейн в освітньому процесі

Blockchain, безперечно, дуже ефективний та потужний інструмент. Це технологія, яка стала основою віртуальної валюти, але швидко стає очевидним, що блокчейн - це більше, ніж просто Bitcoin. Технологія шифрованої книги, що використовує Bitcoin, грунтується на тому, щоб змінити майбутнє багатьох галузей. Нехай це буде охорона здоров'я, фінанси, засоби масової інформації чи уряд, технологія блокчейн призведе до революційних змін у багатьох галузях. Ця технологія, безумовно, стосується й освіти. Не можна заперечувати той факт, що система освіти далеко не у тому стані, у якому вона може бути. Використовуючи цю технологію, можна багато вдосконалити в галузі освіти.

Цифрові сертифікати. У 2017 році MIT (Массачусетський Інститут Технологій) видав віртуальні дипломи своїм випускникам. Студенти отримували їх на свої смартфони. Він відрізнявся від звичайного паперового диплома багатьма способами. На відміну від паперового диплома, блокчейн-диплом ніколи не загубиться. Його також ніколи не можна підробити. Куди б не поїхав студент, його сертифікати завжди залишаються з ним. Необхідність створення традиційного клірингового будинку або університету в якості посередника для видачі стенограм також може бути усунена за допомогою оцифрованих дипломів блокчейн. Тепер, якщо студенти почнуть зберігати всі свої сертифікати та значки на блокчейні, переміщення між університетами стане набагато простішим. Перешкоди, такі як передача кредитів, не перешкоджають їх освітньому процесу. Оскільки блокчейн дуже безпечний, то після додавання даних змінити їх практично неможливо.

Доступна освіта. За даними Forbes, американський студентський борг становить 1,52 трлн дол., І 44 млн позичальників зобов'язані цим боргом. Криза заборгованості студентських позик стала дуже серйозною, і вони

втрачають шкоду в бюджетних роках після закінчення навчання. Все завдяки високій вартості освіти. Технологія може допомогти зробити освіту доступніше для всіх шляхом економії певних коштів. Притаманна гнучкість в його архітектурі дозволяє підтримувати та зберігати записи, документи та цифрові активи без додаткових витрат на інфраструктуру та безпеку.

Поширення МООС (Massive Open Online Courses). Масові відкриті онлайн-курси постійно зростають. Технологія Blockchain може запропонувати цим курсам більшу міру легітимності, надаючи загальнодоступні сертифікати та полегшуючи транзакції для більш легких мікротранзакцій на курсі на основі курсу. Зареєстровано 101 мільйон користувачів масових відкритих онлайн-курсів та 500 облікових даних на базі МООС станом на 2018 рік.

Smart Contract. Блокчейн також може використовуватися для виконання угоди після того, як буде виконано набір інструкцій чи умов. Розумні контракти можуть скоротити оформлення документів у сфері освіти. Наприклад, блокчейн може бути використаний для перевірки відвідування або виконання завдань після того, як буде виконано набір умов.

Якщо казати про певні напрацювання в освітньому напрямку в межах Україні, то можна привести приклад Дрогобицької міської ради, яка запустила в публічне тестування запису до середніх навчальних закладів міста на основі технології Blockchain. Усі заявки батьків щодо запису до навчальних закладів будуть вноситись у базу даних, захищену від фальсифікації. Система побудована так, що надійно захищає персональні дані заявника і дитини. Наразі відділ інформаційних технологій та аналізу виконкому Дрогобицької міськради розробляє дорожню мапу використання технології Blockchain і в інших сферах міста, зокрема реєстрації заяв на

отримання земельних ділянок, у реєстрі комунального майна, договорів тощо. Веб-сервіс називається "Електронна черга дітей в школи" (рис. 1.3).

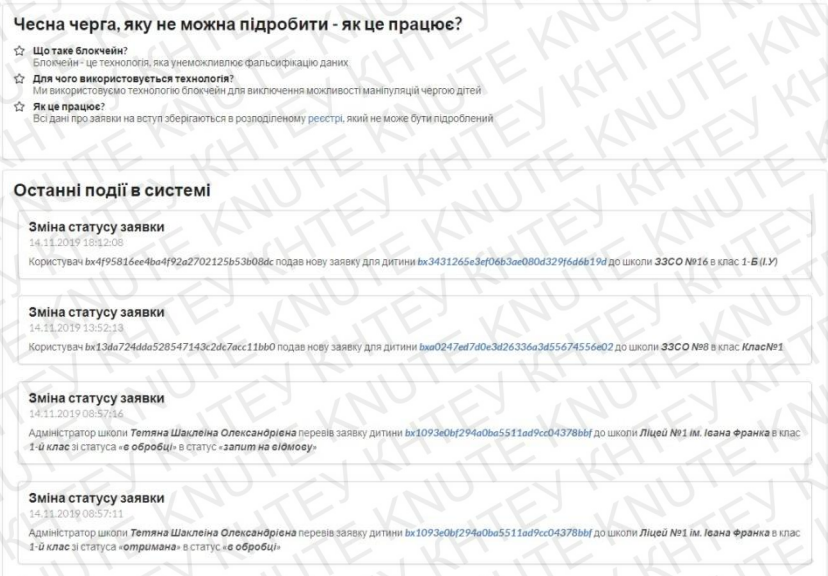

Рис. 1.3 Електронна черга дітей в школи

1.3 Висновки до розділу

Технологія розподіленого реєстру це майбутній стандарт IT, який тільки починає свій шлях. Ця технологія може внести величезний вклад у сферу освіти, оскільки вже сьогодні існує можливість отримати непідробний і єдиний стандартизований сертифікат чи диплом, що гарантує навички та приймається по всьому світу. Активне поширення децентралізованих систем свідчить про те, що дійсно технологія блокчейну може стати найважливішим компонентом цифрового майбутнього та способом пришвидшення темпів глобалізації. В результаті проведеного аналізу можна зробити висновок, що за принципом своєї роботи блокчейн з високою надійністю можна використовувати для побудови інформаційних систем, які призначені для обробки та збереження відкритих даних з метою забезпечення їх цілісності та доступності.

РОЗДІЛ ІІ.
ВИЗНАЧЕННЯ ВИМОГ ТА ІНСТРУМЕНТАРІЮ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Аналіз вимог та визначення задач створення проекту

Працівникам деканату доводиться виконувати величезний обсяг рутинної роботи по обліку студентів, забезпечення освітнього процесу, надання інформації в різні підрозділи закладу вищої освіти. Більш того необхідно здійснювати прогноз успішності студентів з метою ефективного контролю за процесом навчання і своєчасного реагування на можливі негативні результати (низькі показники абсолютної і якісної успішності). Усі ці фактори призводять до неправильного розподілення робочого часу співробітників деканату, гальмуючи і заплутуючи основні процеси діяльності. Перш ніж розроблювати прототип інформаційної системи, необхідно визначити основні вимоги до її роботі. Щодо класифікації вимог було обрано модель Вігерса (рис. 2.1).

Група функціональних вимог:

1. Бізнес-вимоги (Business Requirements) - визначають високорівневі цілі організації або клієнта (споживача) - замовника розроблюваного програмного забезпечення.
2. Вимоги користувача (User Requirements) - описують цілі користувачів системи, які повинні досягатися користувачами за допомогою створюваної системи. Ці вимоги часто представляють у вигляді варіантів використання (Use Cases).

3. Функціональні вимоги (Functional Requirements) - визначають функціональність системи, яка повинна бути створена для надання можливості виконання користувачами своїх обов'язків в рамках бізнес-вимог і в контексті призначених для користувача вимог.
4. Системні вимоги (System Requirements) - класифікуються як складова частина групи функціональних вимог. Описують високорівневі вимоги до ПЗ, що містить кілька або багато взаємопов'язаних підсистем і додатків.

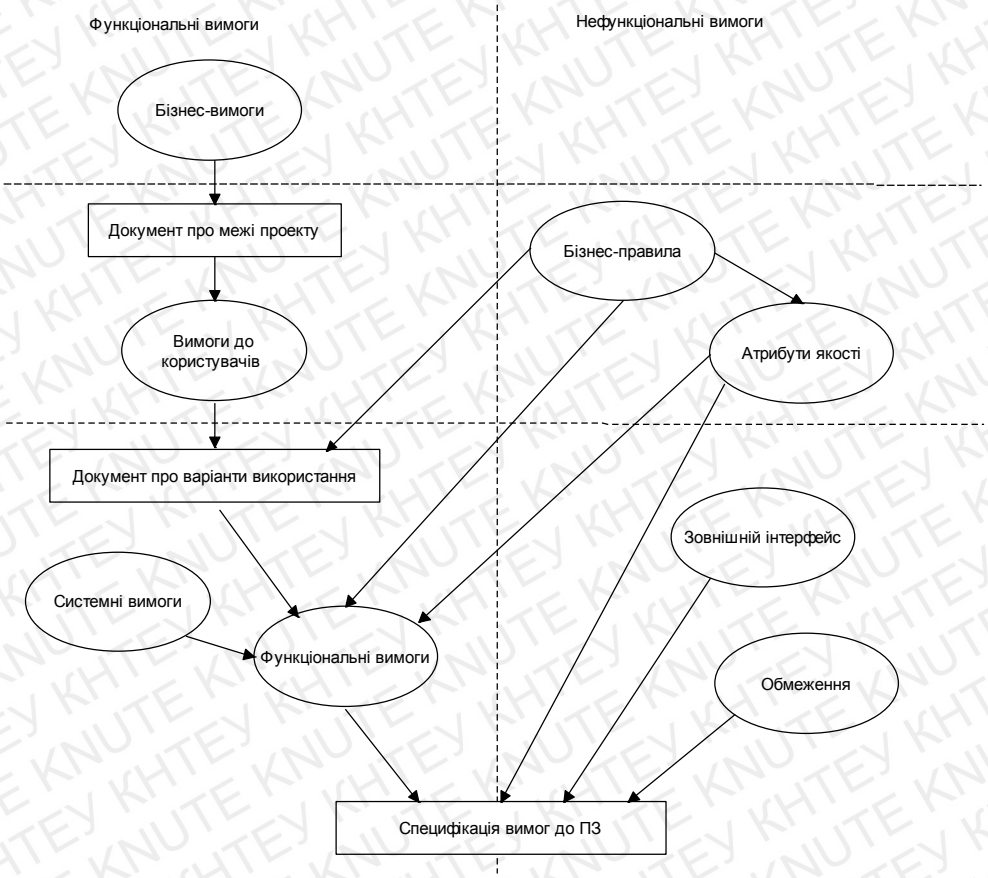

Рис. 2.1 Класифікація вимог за К.Вігерсом.

Група нефункціональних вимог:

1. Бізнес-правила (Business Rules) - включають або пов'язані з корпоративними регламентами, політиками, стандартами. Бізнес-

правила часто визначають розподіл відповідальності в системі, відповідаючи на питання "хто буде здійснювати конкретний варіант, сценарій використання".

2. Зовнішні інтерфейси (External Interfaces) - вимоги до інтерфейсу користувача. Вимоги моють бути направлені на вирішення Бізнес-вимог.
3. Атрибути якості (Quality Attributes) - описують додаткові характеристики продукту в різних станах, важливих для користувачів і розробників. Атрибути стосуються питань цілісності, стійкості.
4. Обмеження (Constraints) - формулювання умов, що модифікують вимоги або набори вимог, звужуючи вибір можливих рішень по їх реалізації. Зокрема, до них можуть ставитися параметри продуктивності, що впливають на вибір платформи реалізації.

Отже, перейдемо до визначення конкретних вимог до системи обліку та аналізу успішності студентів.

Група функціональних вимог:

1. Бізнес-вимоги;

1.1 розробити веб-сервіс для систему обліку студентів;

1.2 адміністративне забезпечення освітнього процесу;

1.3 контроль успішності студентів;

1.4 перевірка якості відвідування студентів;

1.5 розподіл студентів по групах.

2. Вимоги користувача;

2.1 перевірка поточних оцінок по предметах для студентів;

2.2 перегляд карточки студента;

2.3 статистика по відвідуванню занять;

2.4 ведення журналу студентів для викладача;

2.5 перегляд карточки викладача.

3. Функціональні вимоги;

3.1 авторизація студентів та вчителів в "Особистий кабінет" за допомогою логіну та пароля;

3.2 можливість додавання студентів з введенням їх особистих даних;

3.3 розподіл студентів по групах;

3.4 можливість додавання викладачів;

3.5 можливість додавання учбових дисциплін;

3.6 закріплення за певним студентом певних учбових дисциплін з певним навантаження протягом семестру згідно групи та курсу;

3.7 закріплення за викладачами їх фахових учбових дисциплін;

3.8 перегляд студентом особистої карточки;

3.9 студент може переглянути список своїх учбових дисциплін;

3.10 студент може переглянути успішність за певним номером пари дисципліни;

3.11 студент може переглянути загальну кількість пропусків;

3.12 студент можете переглянути кількість пропусків по конкретним дисциплінам;

3.13 студент може переглянути успішність за кожною дисципліною загалом;

3.14 викладач може виставляти пропуски по кожній парі;

3.15 викладач може виставляти оцінки протягом семестру;

3.16 викладач може виставляти екзаменаційну оцінку;

3.17 викладач може продивитись інформацію про себе;

3.18 викладач може дивитись поточний журнал успішності та відвідувань.

4. Системні вимоги;

4.1 система має бути розроблена за допомогою принципів технології блокчейн;

4.2 web-кабінет викладача та студента;

4.3 мова розробки javascript;

4.4 фреймворк розробки angular;

4.5 інформація у системі не може бути змінена чи модифікована;

4.6 super-юзер може переглянути увесь ланцюг блоків (транзакцій).

Група нефункціональних вимог:

5. Бізнес-правила;

5.1 викладач виставляє оцінки студентам;

5.2 викладач відмічає пропуски студентів;

5.3 викладач проставляє екзаменаційні оцінки;

5.4 студент перевіряє свої оцінки та статистику відвідування.

6. Зовнішні інтерфейси;

6.1 gui складається з меню у лівому боці та займає 20% простору;

6.2 центральна (функціональна) частина екрану займає 80% робочого простору;

6.3 відображення усіх даних у табличному виді;

7. Атрибути якості;

7.1 інформація, що була внесена викладачем, не може бути змінена, а може бути тільки відображена;

7.2 студент може бачити тільки свої дані по оцінкам та відвідуванням.

8. Обмеження;

8.1 викладач не має обмежень по кількості дисциплін, що він викладає;

8.2 кожна дисципліна викладається протягом семестру у кількості занять від 5 до 20;

8.3 Кожна група студентів має набір дисциплін в залежності від курсу.

Дані вимоги є певним керівництвом для розробки інформаційної системи, оскільки вони повністю описують перелік функцій, що мають виконуватися системою та окреслюють принципи її функціонування та використовування кінцевими користувачами. У цій роботі ми часто будемо робити посилання на ці вимоги з вказанням номера пункту та підпункту.

2.2 Вибір фреймворку та побудова архітектури

Angular це фреймворк від компанії Google для створення клієнтських додатків. Перш за все він націлений на розробку SPA-рішень (Single Page Application), тобто односторінкових додатків. Наразі дуже активно використовується та розвивається. Згідно SimilarTech.com, який заміряє використання веб-технологій, такий фреймворк як ReactJS в даний час використовується на 112.000 веб-сайтах (при 3.20% зростанні в жовтні 2017 року), а AngularJS (сюди ж входить і Angular) використовується 542.000 сайтами (при зростанні 1.93%) (рис.2.2).

Angular надає таку функціональність, як двостороннє зв'язування, що дозволяє динамічно змінювати дані в одному місці інтерфейсу при зміні даних моделі в іншому, шаблони, маршрутизація (routing) і так далі. Однією з ключових особливостей Angular є те, що він використовує в якості мови програмування TypeScript. Для роботи з Angular необхідно встановити сервер Node.js і пакетний менеджер npm, якщо вони відсутні на робочій машині. Angular - JavaScript фреймворк для фронтенд розробників від Google. Більшість, мабуть, знають різницю між фреймворком та бібліотекою.

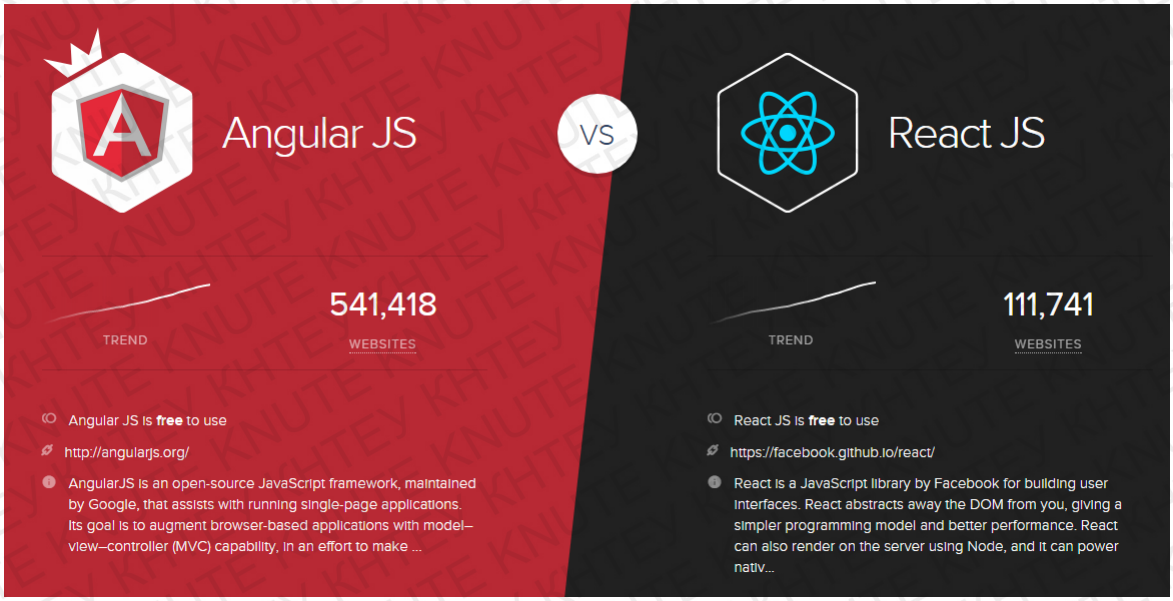

Рис 2.2 Статистика використання Angular на веб-сайтах.

Бібліотеки зазвичай спеціалізуються на вузькій чи конкретній задачі. jQuery наприклад, спеціалізується на роботі з DOM-деревом, створює запити на backend тощо. Але бібліотека не передбачає створення архітектурної системи в цілому. Фреймворк, натомість, призначений для того, щоб побудувати архітектуру усього додатку. Загальна архітектура Angular.JS (рис 2.3).

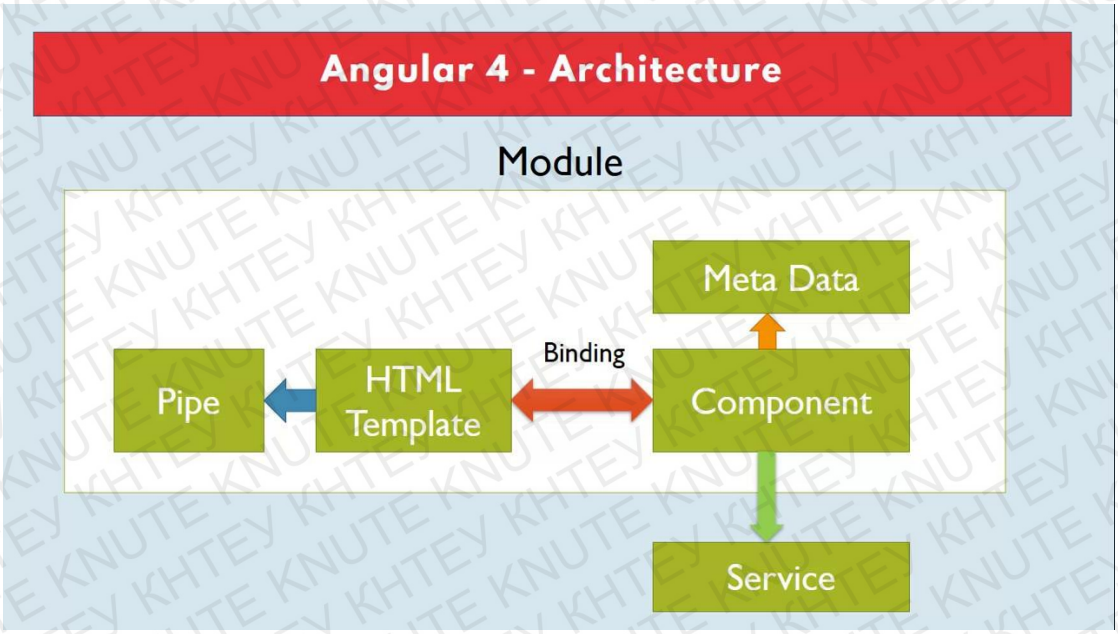

Рис 2.3 Архітектура фреймворку Angular.

Компонент (Component) є одним з основних будівельних блоків у Angular. Додаток може мати більше одного компонента. У звичайному додатку компонент містить файл класу сторінки уявлення HTML, файл класу, який керує поведінкою сторінки HTML, і файл CSS/SCSS для стилізації у HTML. Компонент може бути створений за допомогою @Component decorator, який є частиною модуля @ angular / core.

Команда для імпорту компонента наступна:

- import {Component} from '@ angular / core';

Для створення компонента необхідно прописати наступну команду:

- @Component ({selector: 'greet', template: 'Hello {{name}}!'})

  class Greet {

  name: string = 'World';

  }

Параметр selector - ім'я селектора компонента. З його допомогою розміщується компонент в потрібному місці в HTML. Якщо відкрити index.html, то цей компонент можна знайти тільки в body.

- <App-root></app-root>

templateUrl - посилання на html файл уявлення компонента. Шаблон також можна оголосити інлайн за допомогою властивості template.

styleUrls - масив імен файлів (s) css, на які посилається компонент. Це приватні стилі компоненту, що не торкаються інших частин програми.

Структура компонентів виглядає наступним чином (рис. 2.4).

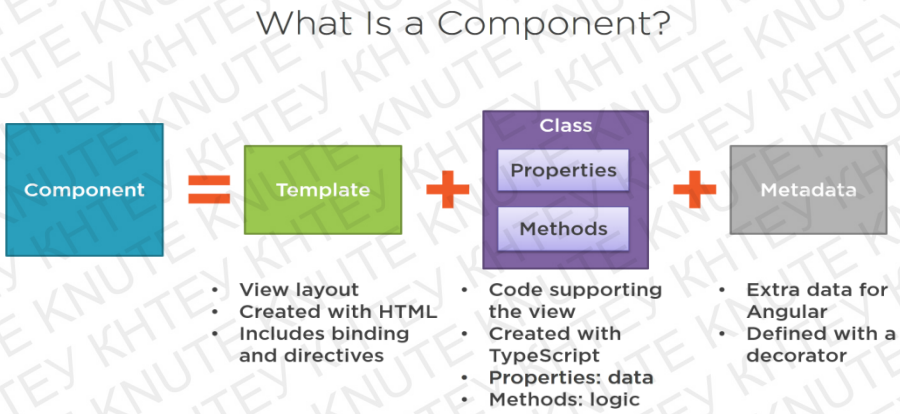

Рис. 2.4 Схема компоненту (Component).

Важливою частиною Angular-додатку є маршрутизація. Маршрутизація дозволяє легко запросити заявки на пропозицію з визначеними ресурсами всередині додатків. Ключовим для роботи маршрутизації є модуль RouterModule, який знаходиться в пакетах @ angular / router. Angular Router - основа платформи Angular. Роутер дозволяє створювати односторінкові застосунки з кількома представленнями та навігацією ними. При роботі з маршрутизацією цей пакет повинен бути вказаний у списку залежностей у файлі package.json (рис. 2.5).

```
{
  "name": "helloapp",
  "version": "1.0.0",
  "description": "First Angular 8 Project",
  "author": "Eugene Popov <metanit.com>",
  "scripts": {
    "dev": "webpack-dev-server --hot --open",
    "build": "webpack"
  },
  "dependencies": {
    "@angular/router": "~8.0.0",
  },
  "devDependencies": {
  }
}
```

Рис. 2.5 Файл package.json з вказанням “dependencies”

Кожен раз, коли натискається посилання або змінюється URL-адреса браузера, маршрутизатор (router) гарантує, що програма реагує відповідно. Для цього router виконує наступні 7 кроків:

- Синтаксичний аналіз: аналіз URL-адреси браузера, до якої користувач хоче перейти;
- Перенаправлення: застосовується перенаправлення (redirect) URL-адреси (якщо визначено);
- Ідентифікування: визначається, який стан маршрутизатора відповідає URL-адресі;
- Охорона: керує охоронцями, які визначені в стані маршрутизатора. Охоронці це сценарії, які запускаються під час завантаження, активації чи деактивації маршруту;
- Вирішення: вирішуються необхідні дані для стану маршрутизатора;
- Активування: активуються компоненти (@Component) для відображення сторінки;
- Керування: керування навігацією та повторення процесу, коли запитується нова URL-адреса.

Для цих кроків використовується мнемічна фраза PRIGRAM, де кожна буква являє собою етап у процесі маршрутизації. Для задання нових URL-адрес їх необхідно вписувати у файл app-routing.module.ts, який містить повний список маршрутів (url-адрес) (рис. 2.6):

```
const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'hero/:id',      component: HeroDetailComponent },
  { path: 'heroes',
    component: HeroListComponent,
    data: { title: 'Heroes List' }
  },
  { path: '',
    redirectTo: '/heroes',
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];
```

Рис. 2.6 Файл app-routing.module.ts з маршрутами

Схема роботи Router у Angular наступна (рис 2.7):

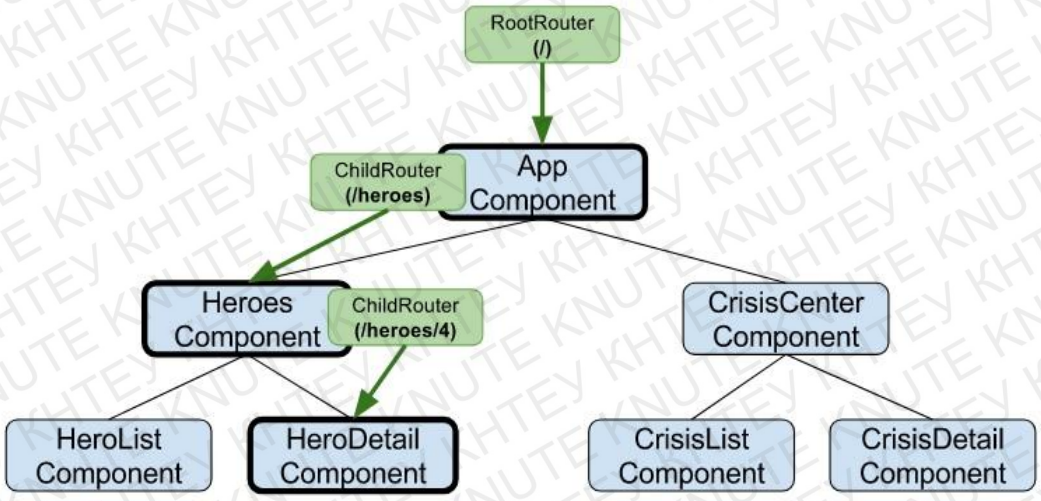

Рис. 2.7 Workflow Router у Angular.

Коли йде перехід до будь-якої заданої URL-адреси, $rootRouter порівнює його конфігурацію маршруту з URL-адресою. Якщо визначник маршруту в конфігурації маршруту розпізнає частину URL-адреси, то компонент, пов'язаний з визначенням маршруту, створюється миттєво і відображається в Outlet'ті. Якщо новий компонент містить власні маршрути, тоді для цього компонента маршрутизації створюється новий маршрутизатор

(ChildRouter). Потім ChildRouter для нового компонента маршрутизації намагається співставити його конфігурацію маршруту з частинами URL-адреси, які вже не були узгоджені попереднім маршрутизатором. Цей процес триває, поки не закінчиться компонент маршрутизації або не закінчиться URL-адреса.

Сервіси в Angular представляють досить широкий спектр класів, які виконують деякі специфічні завдання, наприклад, логування, роботу з даними. На відміну від компонентів і директив сервіси не працюють з уявленнями, тобто з розміткою html, не роблять на неї прямого впливу. Вони виконують строго певне і досить вузьке завдання. Стандартні завдання сервісів:

- Надання даних з додатком. Сервіс може сам зберігати дані в пам'яті, або для отримання даних може звертатися до будь-якого джерела даних, наприклад, до сервера;
- Сервіс може представляти канал взаємодії між окремими компонентами програми.

Сервіс може инкапсулировать бізнес-логіку, різні обчислювальні завдання, завдання по логуванню, які краще виносити з компонентів. Тим самим код компонентів буде зосереджений безпосередньо на роботі з поданням. Крім того, тим самим також можна вирішити проблему повторення коду, якщо буде потрібно виконати одну і ту ж задачу в різних компонентах і класах. Для створення нового сервісу можна використати наступну команду: "ng g service "назва сервису". "g" - команда означає генерацію (generate), а команда "service" уточнює що необхідно сгенерувати саме сервіс. Нижче наведена структура Angular-додатку (рис 2.8). Рисунок наявно демонструє, що сервіси у Angular взаємодіють з вихідним кодом, розширюють та отримують дані.

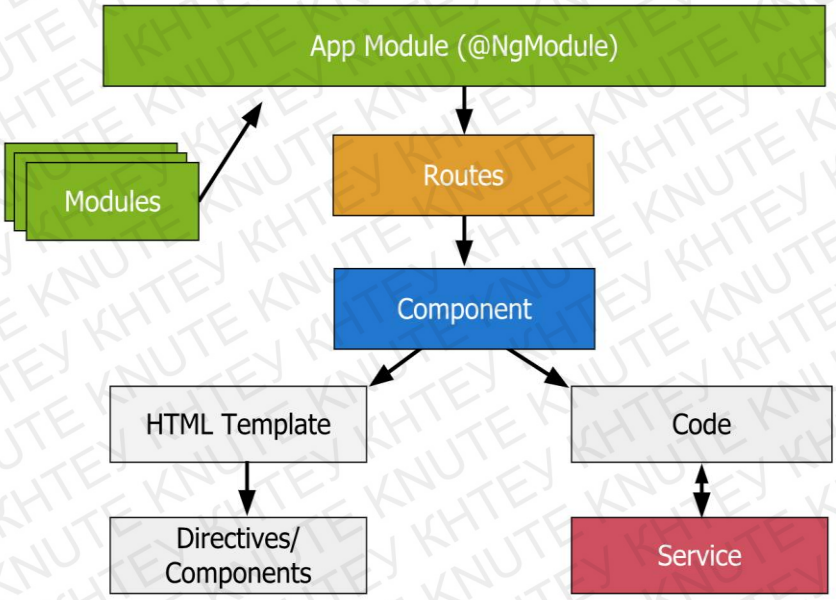

Рис. 2.8 Структура Angular додатку.

2.3 Розробка UML-діаграм

UML - уніфікована мова моделювання (Unified Modeling Language) - це система позначень, яку можна застосовувати для об'єктно-орієнтованого аналізу і проектування. Її можна використовувати для візуалізації, специфікації, конструювання та документування програмних систем. У моїй роботі буде розглянуто три типи UML-діаграм:

- діаграма класів;
- діаграма варіантів використання (use-cases).

Діаграма класів служить для представлення структури системи в термінології класів об'єктно-орієнтованого програмування. Діаграма класів відбиває різні взаємозв'язки між окремими сутностями предметної області, а також описує їх внутрішню структуру (поля, методи). Можна виділити чотири основні сутності, що будуть взаємодіяти між собою у системі:

- викладач;
- студент;

- група студентів;
- дисципліна.

Діаграма класів розроблена за допомогою прикладного програмного забезпечення, описує класи (сутності), що будуть наявні у системі, їх методи та поля та їх звязки між собою. Діаграма класів наведена на рис 2.9.

Ця діаграма класів, навіть у такому спрощеному варіанті, формально виконує пункти "Функціональних вимог" та повністю описує майбутній функціонал.

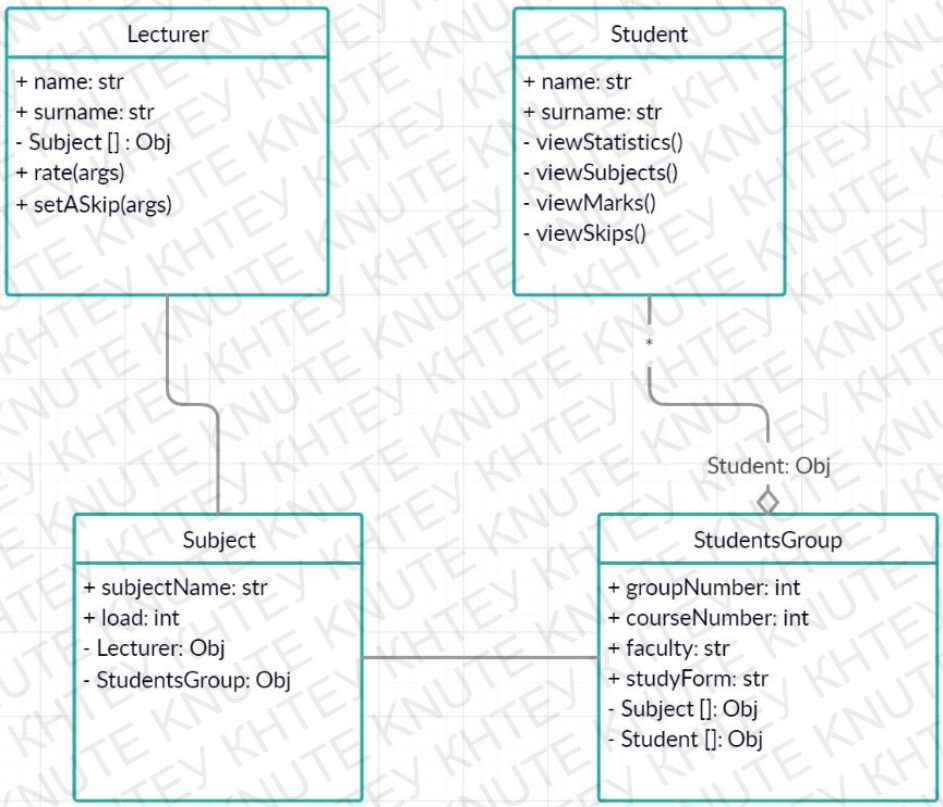

Рис. 2.9 Діаграма класів.

Проектована система представляється у вигляді безлічі сутностей або акторів, що взаємодіють з системою, за допомогою так званих прецедентів (use-cases). При цьому актором (actor) або дійовою особою називається будь-

яка сутність, що взаємодіє з системою ззовні. Іншими словами, кожен варіант використання визначає деякий набір дій, який чинять системою при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізовано взаємодію акторів з системою. Отже перейдемо до розробки USE-case. Перший use-case, що буде розглянутий, це взаємодія super-user (адміністратора) з системою (рис. 2.10). Адміністратор - це найвідповідальніша особа, що відповідає за безпеку системи. Super-user має вручну згенерований private key (приватний ключ) для авторизації. Зокрема кожен користувач має свій private key.

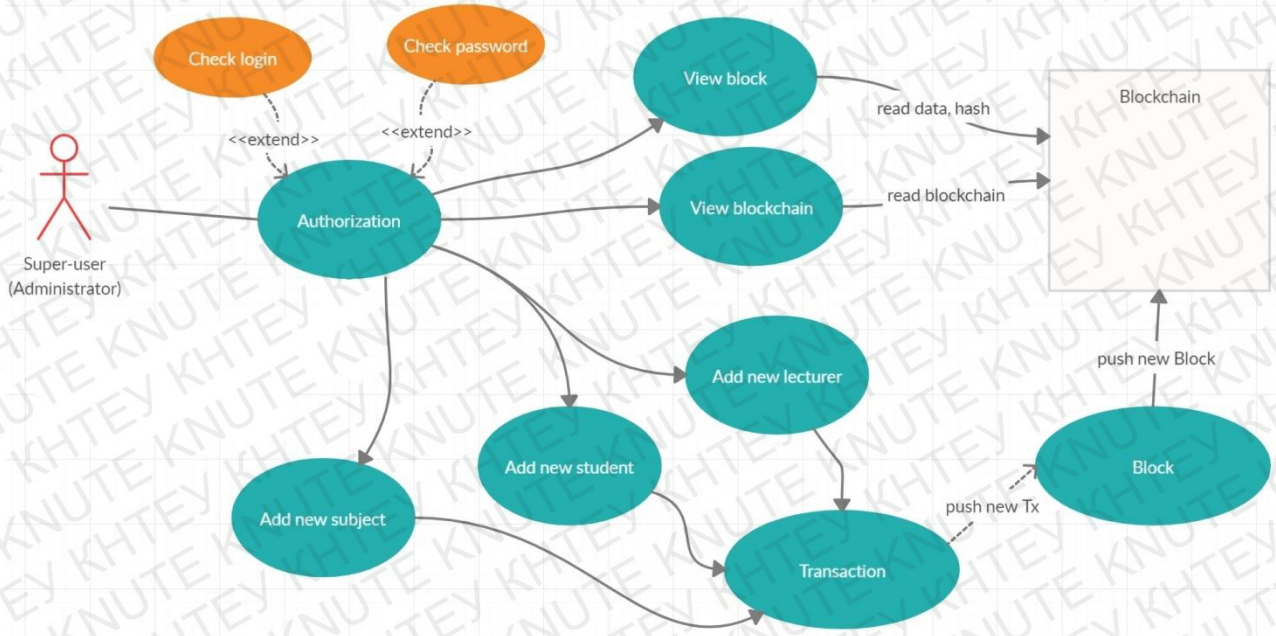

Рис. 2.10 USE-case адміністратора системи.

Коли адміністратор авторизується у систему, йде потрійна перевірка: перевірка наявного логіну, перевірка коректності паролю та перевірка введеного приватного ключа. Super-user може виконувати основні системні функції, такі як перегляд блокчейну (ланцюга блоків), перегляд вмісту конкретного блоку, додати нового студента, викладача чи додати нову дисципліну. Також розглянемо use-case взаємодії викладача та майбутнього веб-сервісу(рис 2.11).

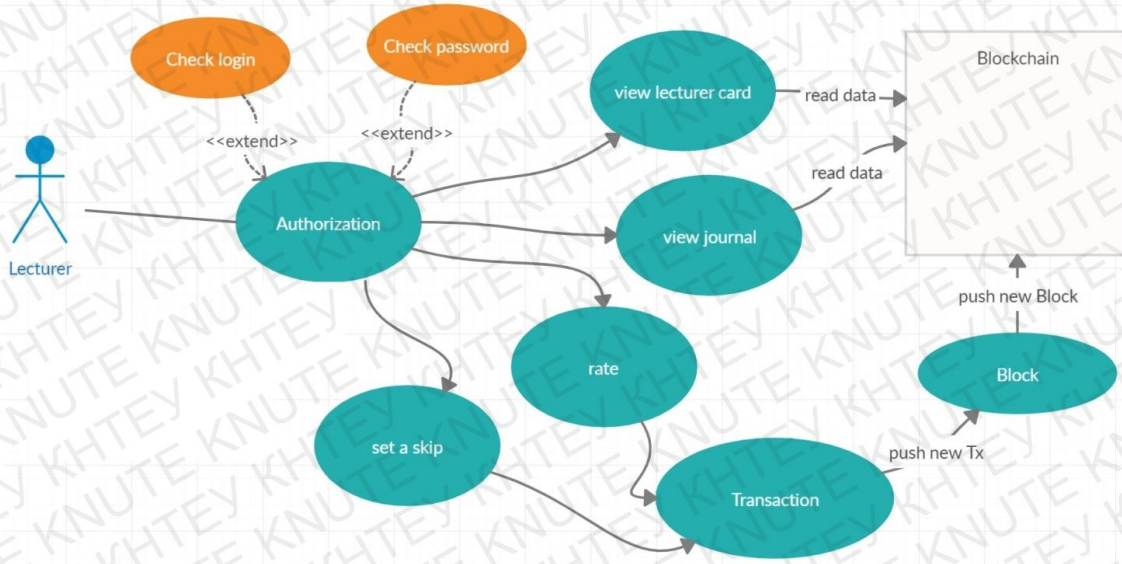

Рис. 2.11 USE-case викладача.

Наданий use-case (рис 2.11) формально реалізує та покриває наступні вимоги: 2.4, 2.5, 3.1, 3.14-3.18. Після успішної авторизації за логіном та паролем, викладач може виставити пропуски студентам що були відсутні за поточним номером заняття, може виставити певний бал за роботу в аудиторії чи за виконання домашнього завдання, також може відкрити поточний журнал успішності студентів, де відображений увесь список поточних балів, пропусків за усіма студентами по групах. Викладач має змогу відкрити свою картку викладача, де буде відображена інформація про викладача. Після завершення семестру та проведення екзаменаційної сесії викладач виставляє оцінки кожному студенту по групах.

Останній use-case, що буде наведений, це взаємодія студента університету з веб-сервісом (рис 2.12):

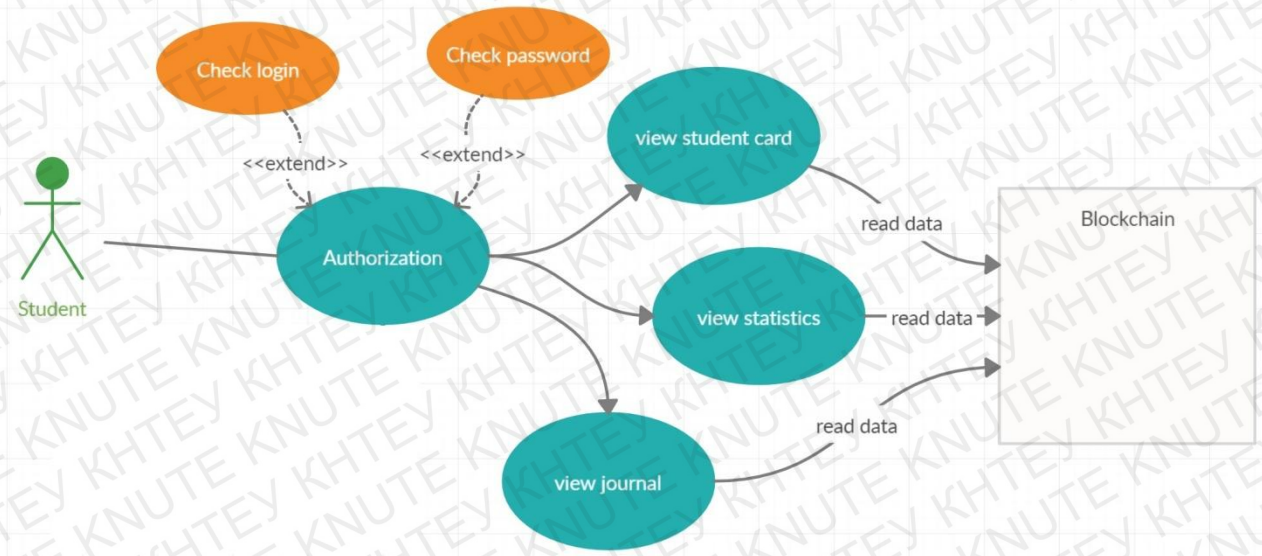

Рис. 2.12. USE-case студента.

Наданий use-case(рис 2.12) формально реалізує наступні вимоги: 2.1-2.3, 3.1, 3.8-3.13. Після успішної авторизації, студент може відкрити студентську картку, де буде детальна інформація по студенту, сформувати поточну статистику по оцінках та відвідуваннях (в кінці семестру також може дізнатись оцінку за екзамен). І останнє, що може зробити студент-користувач, це відкрити поточний журнал і сформувати усі свої оцінки та пропуски по всім предметах протягом навчальних років в університеті. Студент не добавляє жодної інформації у блокчейн, він її тільки "зчитує", оскільки інший функціонал не визначений вимогами до поточної інформаційної системи.

2.4 Моделювання блокчейну

Моделюючи блокчейн, не можна не згадати такі поняття як "Proof of Work" та "Proof of Stake". Кожен з цих принципів може бути реалізованим як механізм валідації транзакцій у блокчейні. Дослівно "Proof of Work" перекладається як "доказ роботи" або доказ виконаної роботи - за роботу приймається об'єм обчислювальних операцій обладнання. Proof of Work - це механізм перевірки того, що робота (майнінг) була проведена. Цей механізм

є загальним випадком вирішення задачі Візантійських генералів. В умовах взаємодії віддалених учасників мережі, частина з яких можуть бути зловмисниками, необхідно знайти виграшну для всіх стратегію. Процес вирішення цього завдання можна розглядати через призму змагальних моделей. Механізм PoW використовується в мережі Bitcoin для генерації блоку і безпеки усього блокчейну. У цих блоках і міститься хеш-функція, сума якої завжди менше target (поставленої мети). Це показує або доказує(proof), що необхідні розрахунки (work) для пошуку блоку були проведені і дає сигнал до того, що блок можна записати в загальну ланцюжок (блокчейн). Основним алгоритмом хешування є SHA-256.

У Proof of Stake (PoS) в якості ресурсу використовується розмір частки (Stake), який і визначає, хто з вузлів в результаті знайде блок і отримає винагороду. Майнінг (підтвердження транзакцій) відбувається за рахунок наявності монет на гаманці, і чим їх більше - тим вище нагорода. Вузол, який одержує винагороду за утримання певної частки (stake) ще називають мастернодой. PoS має наступні значущі переваги:

- цей механізм консенсусу в мережі вимагає менше ресурсів ніж PoW;
- класичною атаки 51% в блокчейні з PoS бути не може - тому що обчислювальні потужності не грають ролі при ранжируванні нод;
- навіть якщо атака відбудеться, то робота блокчейну буде порушена і буде складно отримати з цього вигоду;
- потенційна атака може статися тільки в тому випадку, якщо в руках одного вузла зосереджено 51% всіх монет - а це дуже і дуже дорого.

Але у випадку випускно-кваліфікаційного проекту не один з цих принципів не буде реалізований, підтвердження транзакцій буде відбуватися з перевіркою лише однією функцією. Кожна транзакція також буде підписана відправником своїм приватним ключем, разом з інформацією у транзакції і тому у системі не буде змоги підробити будь-яку транзакцію(рис. 2.13).

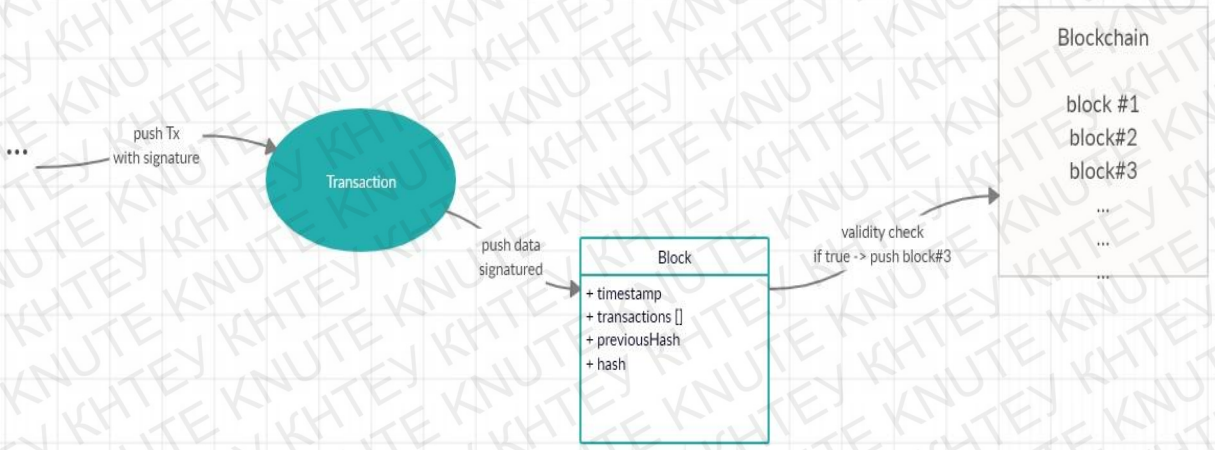

Рис. 2.13. Схема роботи блокчейну та додавання блоку у ланцюг блоків.

Також буде виключений такий момент як несанкціоноване редагування інформації. Якщо у системі буде така спроба, валідність блоку перевіряється перед додаванням у блокчейн. Оскільки у кожному блоці записується хєш-код попереднього та поточного блоку, перед додаванням нового блоку йде перевірка поточних хєш-кодів усіх блоків(рис. 2.14).

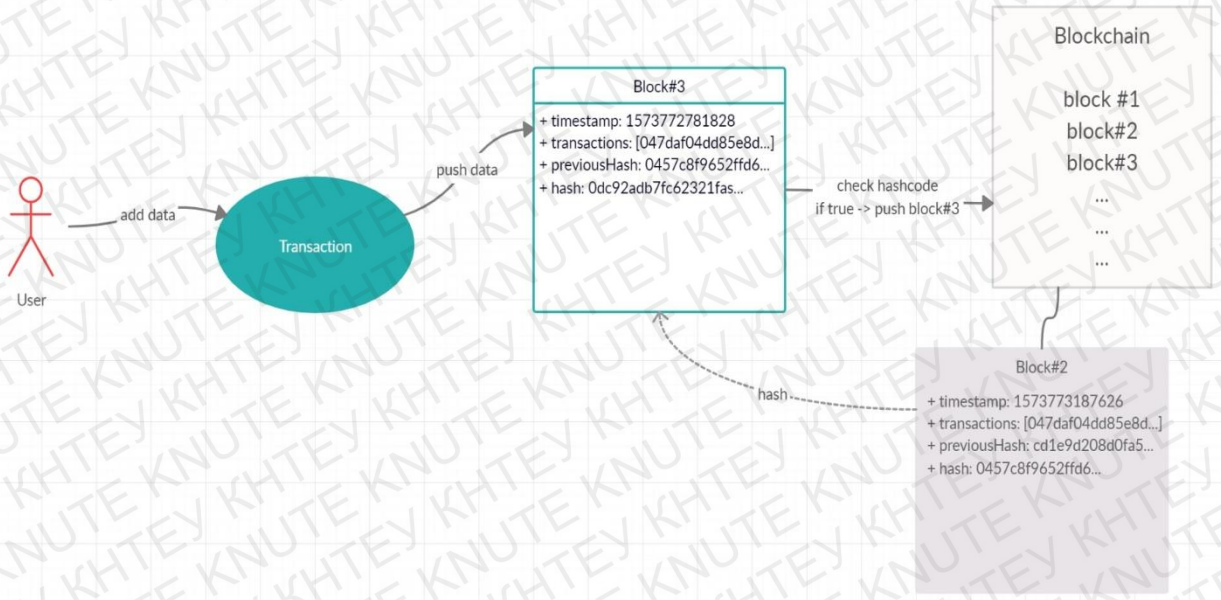

Рис. 2.14. Демонстрація хеш-коду блоків.

Якщо хєш-код хоч одного блоку не співпадає - система зазнала спроби несанкціонованої модифікації. Навіщо йде перевірка хєш-кодів? Будь-яку інформацію можна закодувати у SHA-256 і отримати унікальний хеш-код.

Якщо у блоках зміниться будь-яка інформація, тоді зміниться і їх хеш-код. Розглянемо наступну ситуацію (рис.2.15).

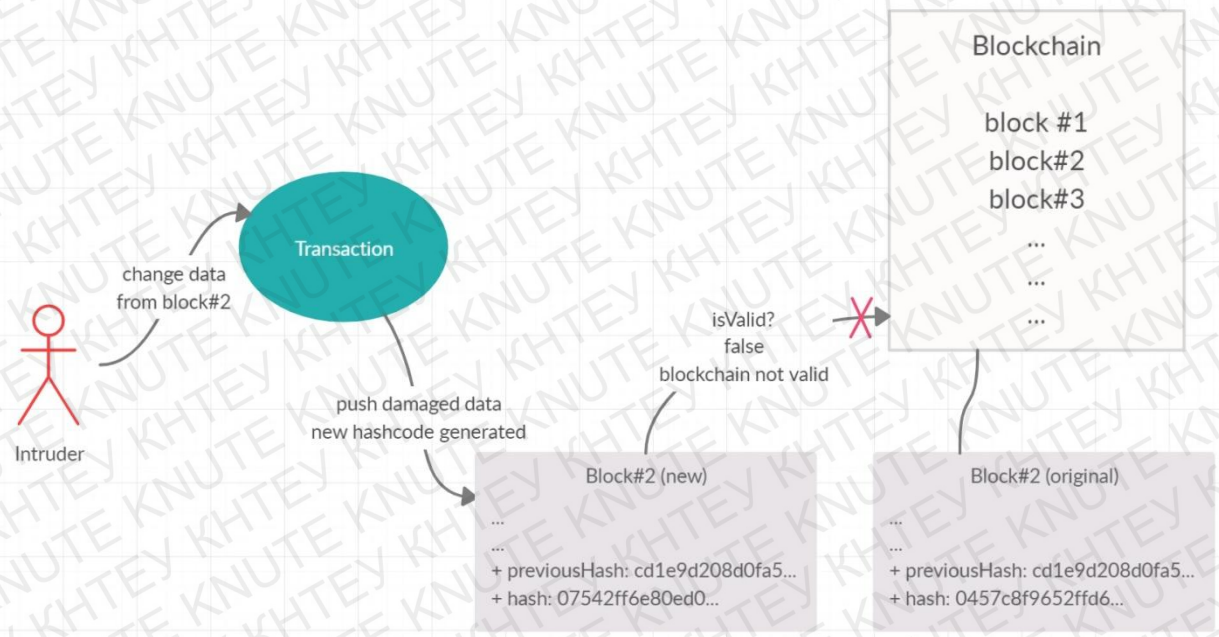

Рис. 2.15 Зловмисник намагається змінити дані у блокчейні.

Якщо зловмисник якимось чином зможе відправити пошкоджені дані на блокчейн, згідно (рис.2.15) хеш-код блоку зміниться. При додаванні блоку до блокчейну у системи виникне питання: чи коректний блок буде добавлено? Система зробить порівняння двух хеш-кодів (07542ff6e80ed0... та 0457c8f9652ffd6...), зрозуміє що вони різні, видасть помилку "Blockchain not valid" та не зареєструє цей блок. Саме за таким простим принципом функціонує технологія блокчейн.

2.5. Висновки до розділу 2

У результаті роботи над розділом було виконано наступні пункти:

- проаналізовано та узгоджено вимоги до майбутньої інформаційної системи обліку та аналізу успішності студентів;
- вимоги класифіковано за методологією Карла Вігерса з розділенням вимог на функціональні та нефункціональні;

- методологією розробки обрано RUP, що орієнтований на створення і підтримання моделей;
- фреймворком створення інформаційної системи обрано Angular.JS та наведено статистику, що свідчить про велику надійність та широке використання даного фреймворку;
- наведено схематичну архітектуру додатку Angular та описано роботу основного методу роботи з представленням даних у Angular - компонентування (робота з Components);
- проведено декомпозицію систему та визначено основні сутності майбутньої інформаційної системи;
- створено USE-case діаграми для основних трьох сутностей: адміністратор, викладач та студент;
- визначено функціональні можливості у залежності від ролі користувача;
- змодельовано блокчейн-систему та розібрано принципи PoW та PoS;
- продемонстровано схему роботи ланцюжкової моделі блокчейну;
- обгрунтовано фундаментальність використання хєш-кодів та принцип їх генерації.

РОЗДІЛ ІІІ.
РОЗРОБКА СИСТЕМИ ОБЛІКУ ТА АНАЛІЗУ СТУДЕНТІВ ЗАСОБАМИ БЛОКЧЕЙН ТА ANGULAR.JS

3.1 Розробка програмного продукту

Перш ніж розпочати роботу над проектом, необхідно завантажити та встановити програмну платформу (веб-сервер) Node.JS (рис. 3.1).

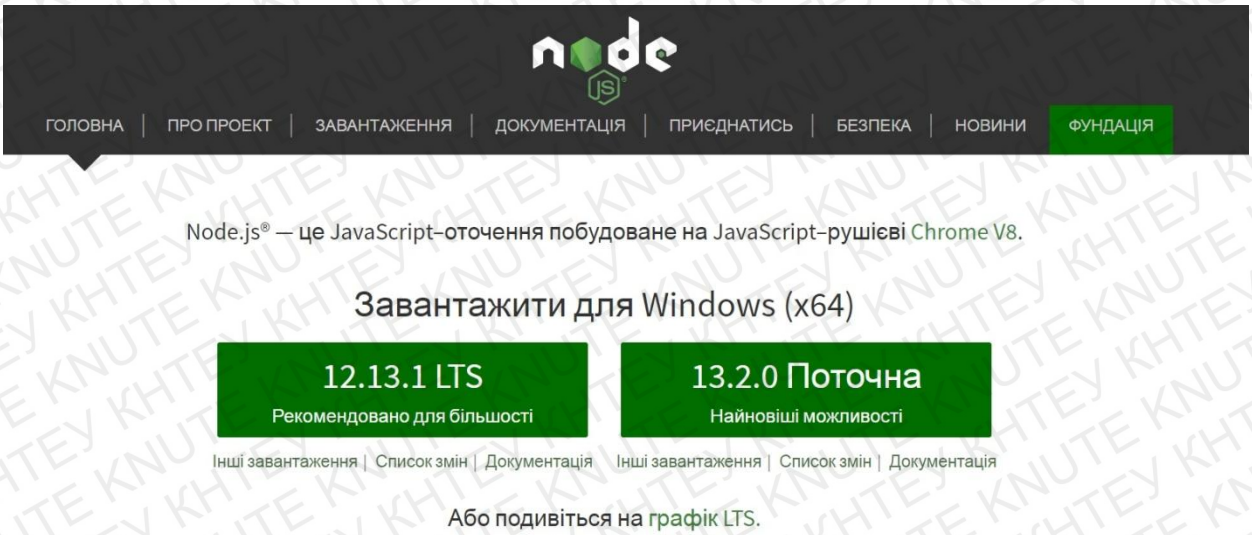

Рис. 3.1. Сайт https://nodejs.org/

Після встановлення Node.JS, необхідно встановити Angular.JS. Задля цього необхідно перейти у командну строку (cmd.exe у Windows) та прописати наступну команду для встановлення Angular (рис 3.2).

```
npm install -g @angular/cli
```

Рис. 3.2 Команда для встановлення Angular.JS

Якщо інсталяція пройшла успішно, у консолі отримаємо наступне повідомлення "updated 1 package" та поточну версію Angular.JS (рис. 3.3):

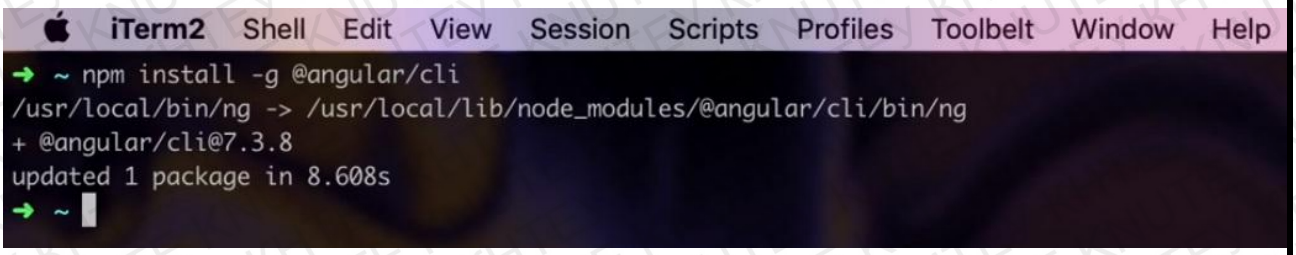

Рис. 3.3 Успішне встановлення Angular.JS

Тепер необхідно створити структуру проекту Angular. Для цьому ми створюємо проект у інтегрованому середовищі розробки (IDE) JetBrains IDEA (рис. 3.4):

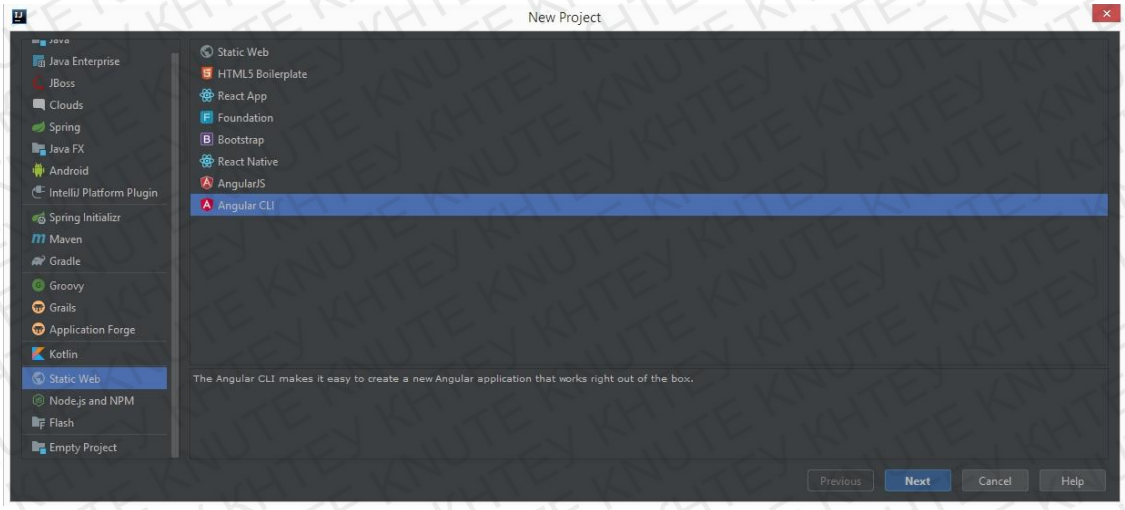

Рис. 3.4 Створення нового проекту Angular.

Новий проект отримає наступну структуру файлів (рис. 3.5):

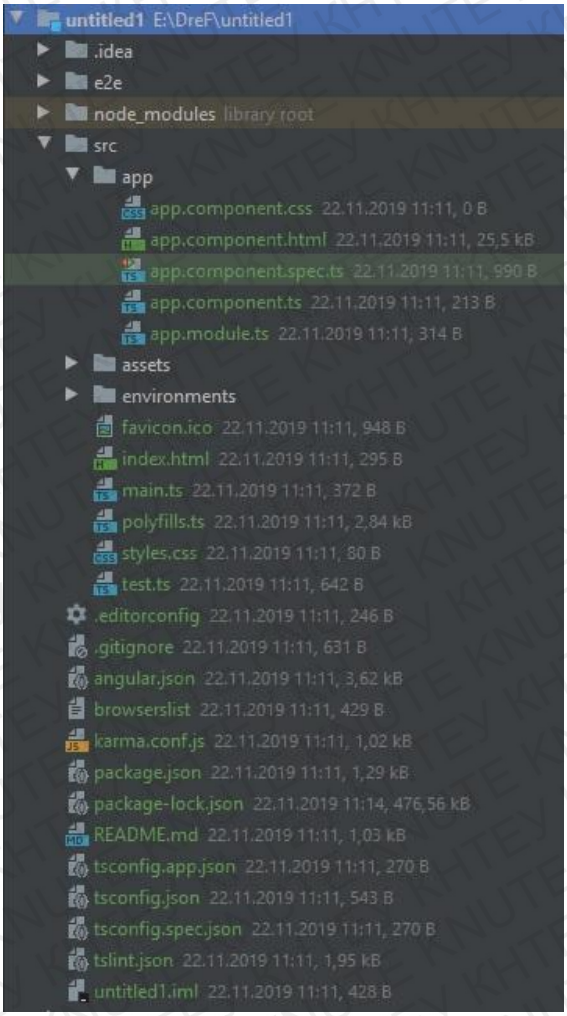

Рис. 3.5 Файлова ієрархія проекту Angular.

Папка "src/app/" містить основний вихідний код веб-сервісу, файл "index.html" містить структуру HTML-документу для усіх сторінок (рис. 3.6).

index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Untitled</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Рис.3.6 Файл index.html

Файл "index.html" містить у тілі тега <body> коріневий модуль (тег) <app-root></app-root>. Отже теперь у тілі тегу <body> буде наш frontend та генерація HTML відображення компонентів (Components).

Використання додатку Angular починається з файлу mait.ts. Там міститься платформна логіка. Файл styles.css служить глобальним файлом стилів, які визначають css-правила для усіх компонентів. Коли генеруються компоненти, також генеруються файли стилів в розширенням .css, в них можно прописувати певні локальні стилі, які актуальні саме для цього компоненту, тобто не прописувати їх в глобальний стиль, а розділити їх по сторінкам (рис. 3.7).

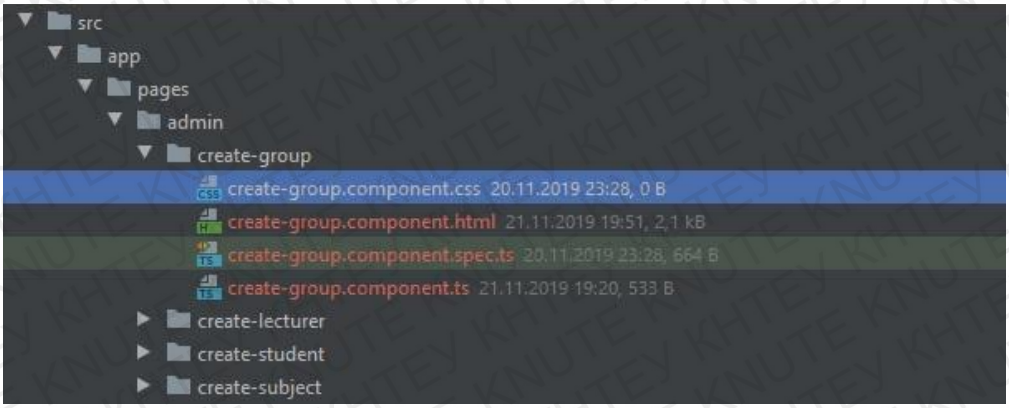

Рис. 3.7 Структура компоненту.

Виходячи з рис. 3.7, можна описати структуру кожного компонента системи, котрий генерується:

- .css файл стилів для конкретної сторінки;
- .html файл з HTML-структурою конкретної сторінки;
- .spec.ts файл специфікації - це одиночні Unit-тести вихідного коду. Конвенція Angular додатків полягає у тому, щоб мати .spec.ts файл для кого .ts файлу. Вони виконуються за допомогою Jasmine (javascript framework для тестування), коли ми вводимо "ng test" у командний рядок для тестування;
- .ts файли Typescript з вихідним кодом (розширенний Javascript).

Графічний інтерфейс вирішено будувати за принципами Material Design - стандарту, що був розроблений Google в 2014 році (рис. 3.8).

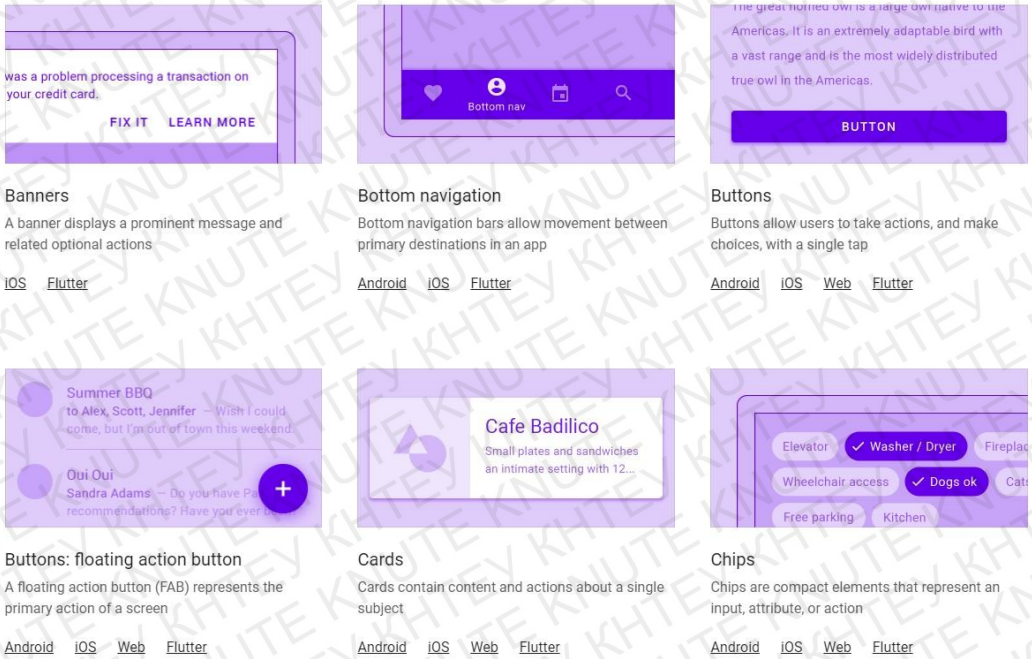

Рис. 3.8 Приклад Material Design.

Material Design спрощує розробникам настройку UI, зберігаючи при цьому зручний інтерфейс додатків. З Material Design ми отримаємо добре організований формат і гнучкість. У Angular є власна бібліотека Angular Material, що складається з набору попередньо встановлених компонентів Angular. На відміну від Bootstrap, який надає компоненти, які ви можете використовувати будь-яким способом, Angular Material прагне забезпечити розширений і послідовний інтерфейс. У той же час він дає можливість контролювати, як поводяться різні компоненти. Для встановлення та додавання Angular Material до нашого проекту необхідно прописати наступну команду у командному рядку: " ng add @angular/material ". Ця команда додатково виконає такі конфігурації:

- додасть залежність проекту до package.json
- додасть шрифт Roboto до свого index.html

- додасть favicon.ico Angular Material до index.html
- додасть кілька глобальних стилів CSS: видалить відступи (margins) з тегу (<body>), встановить висоту: 100% для <html> та <body>, встановить Roboto як шрифт за замовчуванням.

Тепер, коли ми підключили Angular Material, кожен елемент, котрий нам необхідний, треба підключати у файл "app.module.ts" наступним чином (рис.3.9):

```
import { MatSliderModule } from '@angular/material/slider';
…
@NgModule ({....
  imports: [...,
  MatSliderModule,
…]
})
```

Рис. 3.9 Приклад підключення елементу "слайдер".

Перевірити працездатність елементу можна додавши його на розмітку HTML наступного формату (рис. 3.10):

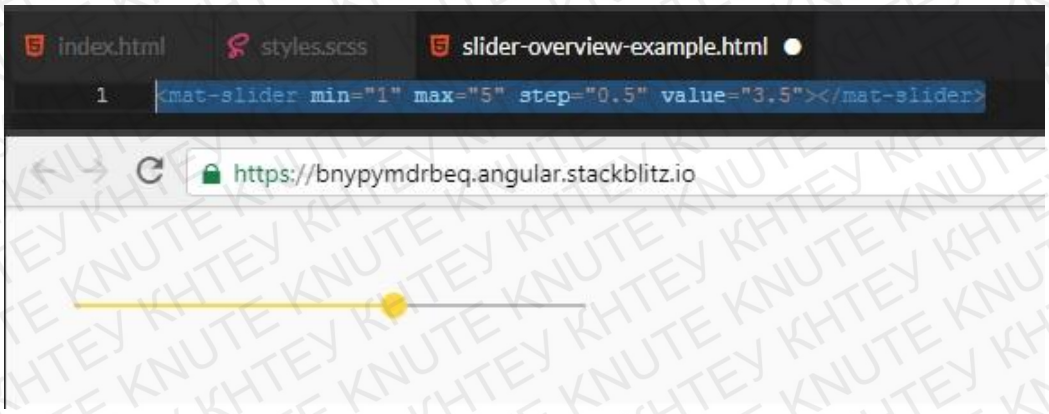

Рис. 3.10 Тестування елементу "слайдер".

Після того, як було обрано принципи побудови інтерфейсу, було розроблено файлову ієрархію проекту (рис. 3.11).

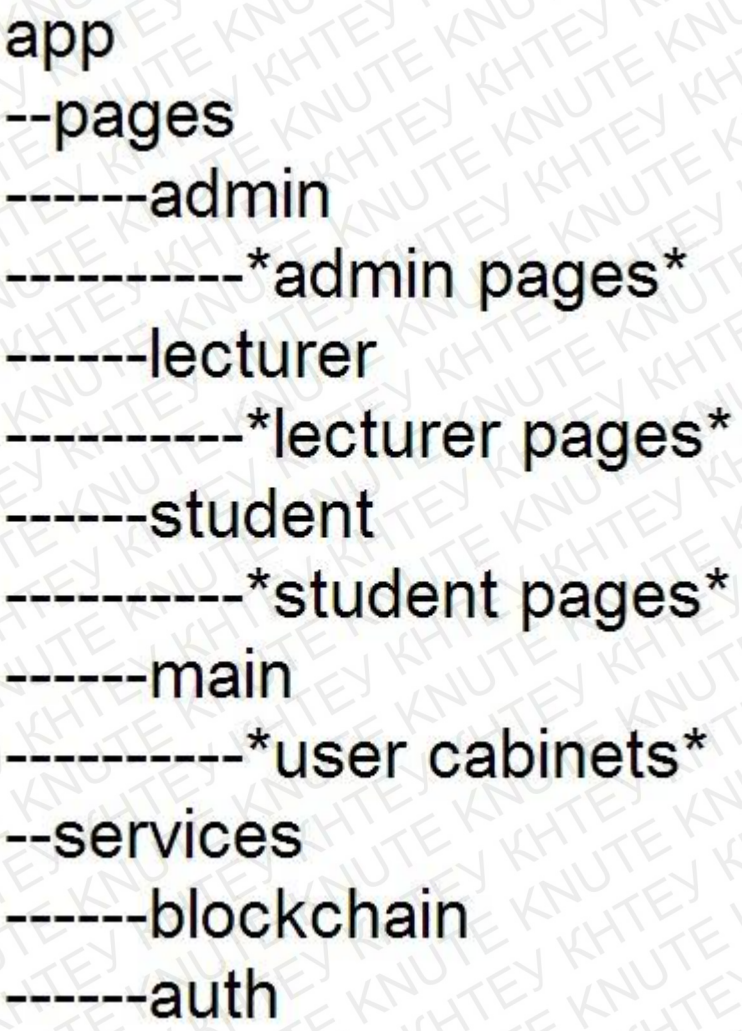

Рис. 3.11 Файлова структура проекту.

Папка "pages" містить створені компоненти (Components) по трьох типах користувача, папка "main" містить код та представлення кабінетів користувачів. Папка "services" містить вихідний код блокчейну та системи авторизації.

Наступним етапом стало створення системи авторизації. Було вирішено використати напрацьоване рішення через сервіс Google. Авторизація розроблена за допомогою Google Firebase з авторизацією через Google аккаунт. У файл environments/enviroment.ts було додано наступний код (рис. 3.12):

```
<script src="https://www.gstatic.com/firebasejs/5.3.1/firebase.js"></script>
<script>
  // Initialize Firebase
  var config = {
    apiKey: "AIzaSyBvqLy4jAWpPNokwxBo24ceMVKoj8aYl2U",
    authDomain: "ang-firebase-aee83.firebaseapp.com",
    databaseURL: "https://ang-firebase-aee83.firebaseio.com",
    projectId: "ang-firebase-aee83",
    storageBucket: "ang-firebase-aee83.appspot.com",
    messagingSenderId: "173586696760"
  };
  firebase.initializeApp(config);
</script>
```

Рис. 3.12 Скрипт Google Firebase.

У кабінеті користувача Firebase є база даних користувачів, які будуть мати доступ до нашої системи (рис. 3.13).

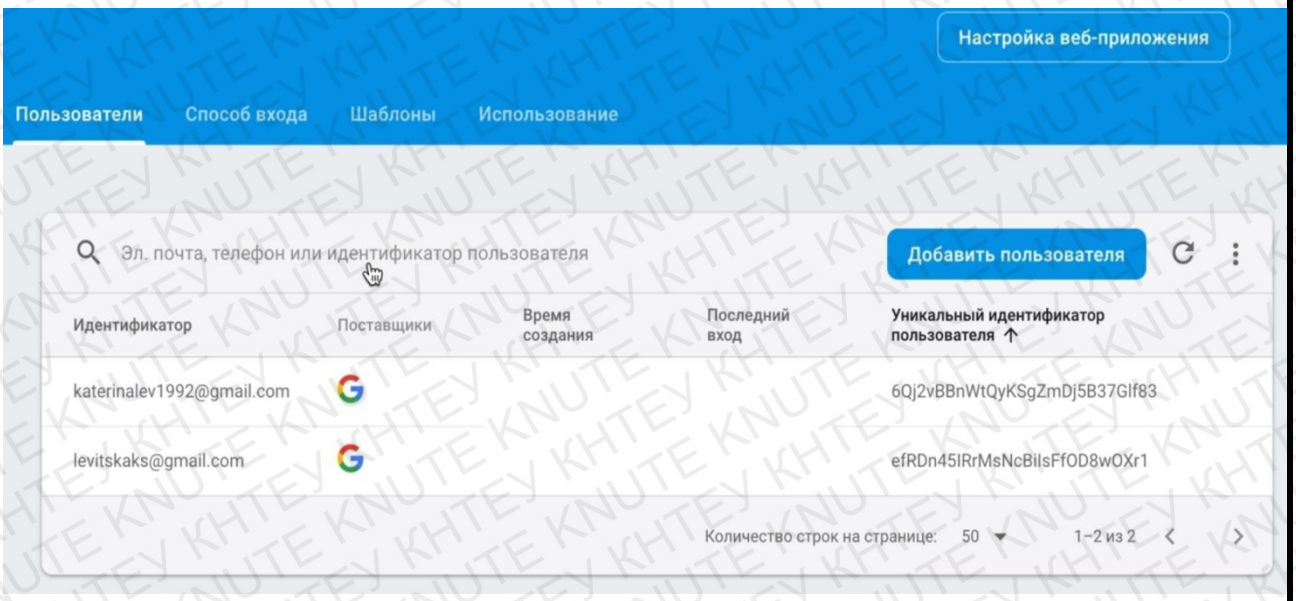

Рис. 3.13 База даних користувачів.

Кожен користувач отримує унікальний ідентифікатор користувача, а після створення проекту ми отримуємо ключ API. Авторизація у систему можлива через звичайний емейл та пароль, чи через Google Аккаунт. Frontend сторінки виглядає наступним чином (рис. 3.14):

Рис. 3.14 Форма авторизації.

Настала черга розробки кабінетів користувачів та розроблення функціоналу. Для відображення усіх головних сторінок було створено три їх прототипи для кожної ролі користувача (адміністратор, викладач та студент). Кабінет super-user (адміністратора) виглядає наступним чином (рис.3.15):

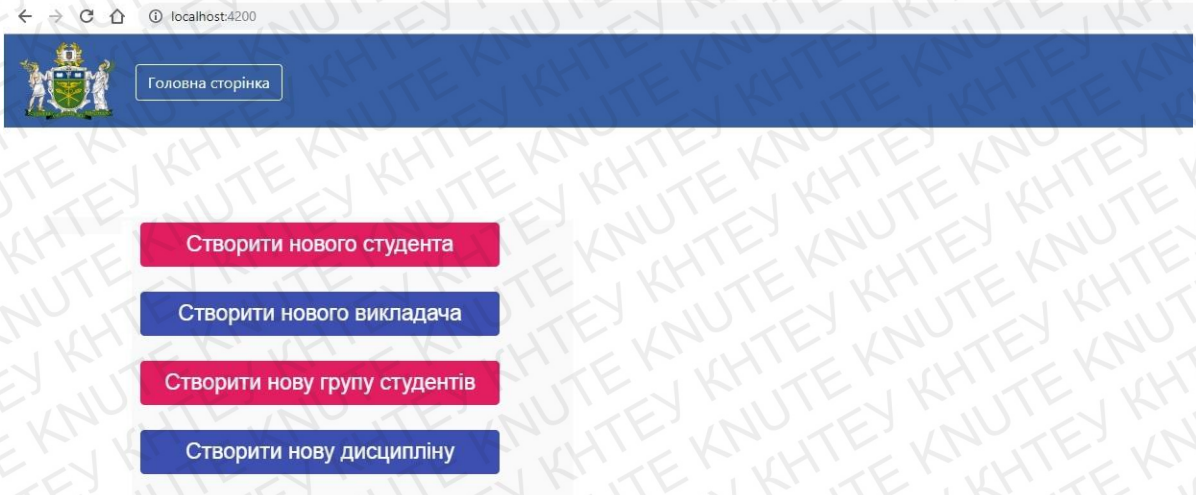

Рис. 3.15 Головна сторінка кабінету адміністратора.

Кабінет викладача (рис. 3.16):

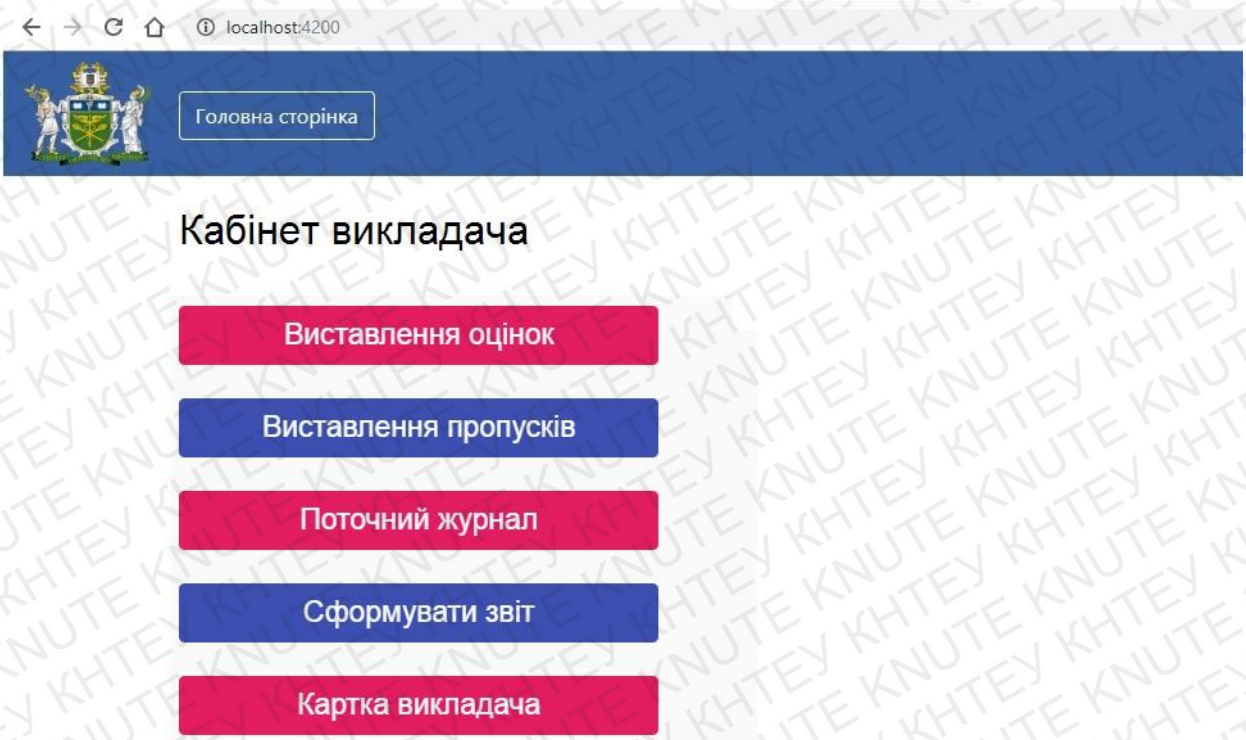

Рис. 3.16 Головна сторінка кабінету викладача.

Кабінет студента (рис.3.17):

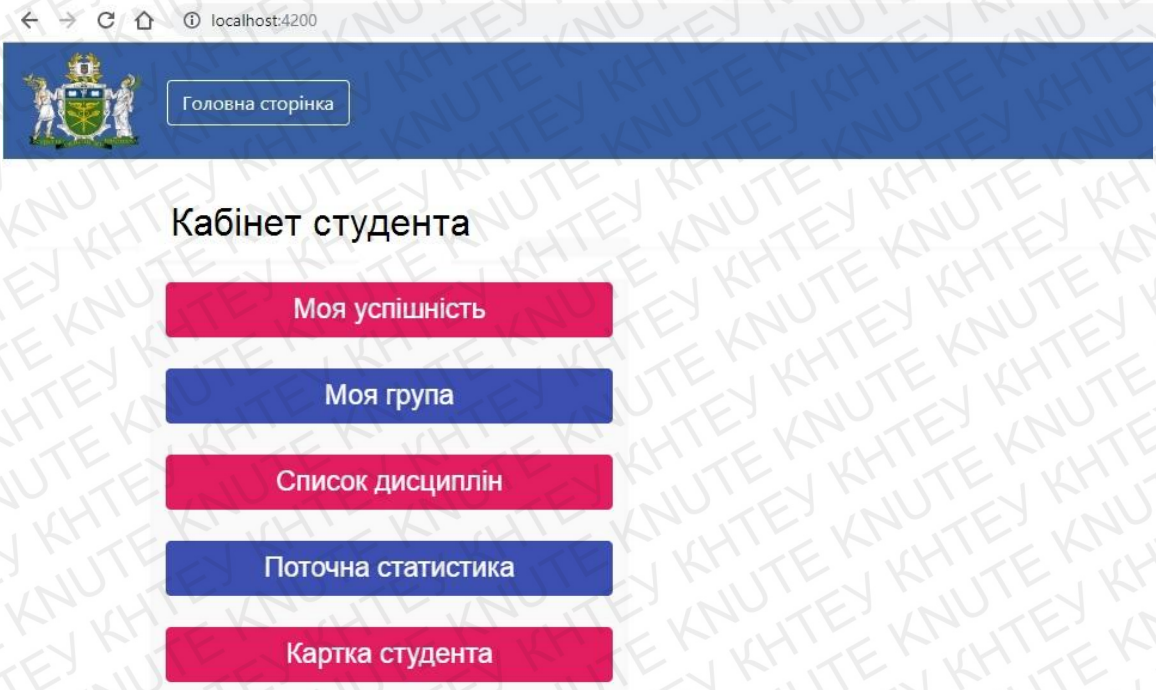

Рис. 3.17 Головна сторінка кабінету студента.

У кабінеті викладача вкладки "Виставлення оцінок", "Виставлення пропусків" мають наступну логіку: спочатку йде вибір дисципліни, далі вибір групи, у якої викладає даний викладач, з'являється табличний список студентів, де за номером пари можна проставляти пропуски, виставляти оцінки. Коли всі оцінки виставлені разблокується поле "Оцінка екзаменаційної сесії". Декілька скріншотів функціоналу наведено нижче (рис. 3.18):

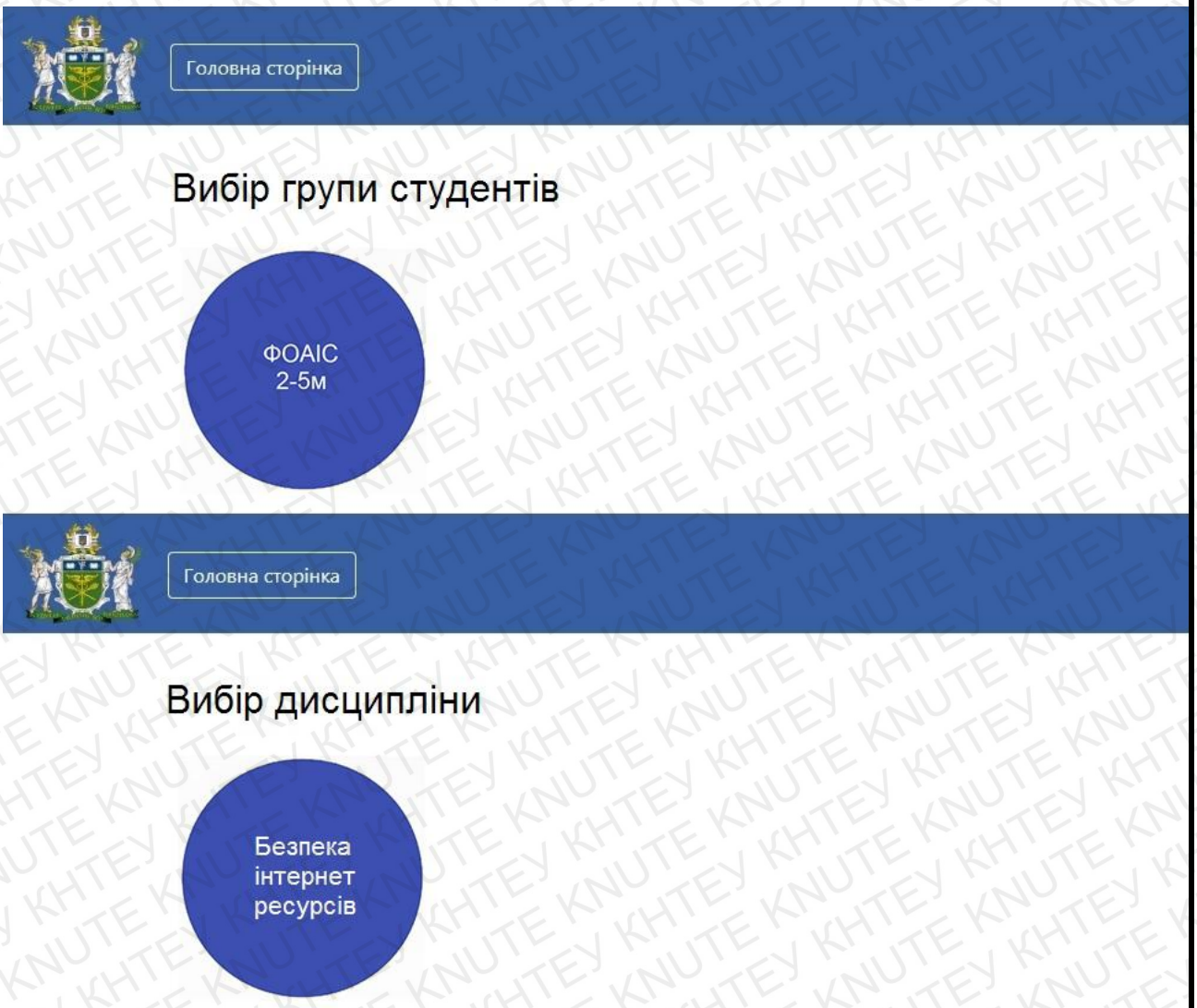

Рис. 3.18 Режим вибору групи і вибір дисципліни.

Виходячи з обраної групи з'являється список дисциплін, які викладаються викладачем для цієї групи. Для прикладу була взята група

ФОАІС 2-5м, дисципліна "Безпека інтернет ресурсів" та викладач Савченко Тетяна Віталіївна. Після вибору дисципліни йде перехід до таблиці студентів з можливістю виставляти оцінки та пропуски (рис. 3.19):

Виставлення оцінок та пропусків				
"Безпека інтернет ресурсів", група ФОАІС 2-5м				
Номер пари	1	2	3	4
ПІБ студента	Оцінка/відвідування			
Александренко Андрій Анатолійович	5	4		
	✓	✓		
Архіпчук Броніслав Павлович	4	3		
	✓	✓		
Бердар Костянтин Анатолійович	4	3		
	✓	✓		
Білецький Вадим Анатолійович	2	3		
	✓	✓		
Вишняк Ярослав Олексійович	0	0		
	✗	✗		
Вікторов Владислав Віталійович	5	0		
	✓	✗		
Віон-Дюрі Ярослав Дмитрович	0	0		
	✗	✗		
Гасімов Орхан Хемідага-огли	4	0		
	✓	✗		
Гелетуха Олександр Аркадійович	4	4		
	✓	✓		
Гулаков Владислав Васильйович	3	3		
	✓	✓		
Деремешко Максим Михайлович	2	2		
	✓	✓		
Залевський Богдан Павлович	2	2		
	✓	✓		
Івлєв Ігор Олександрович	2	2		
	✓	✓		
Кінзерський Костянтин Юрійович	3	4		
	✓	✓		
Кубишев Дмитро Вячеславович	0	0		
	✗	✗		
Кутковий Нікіта Георгійович	4	4		
	✓	✓		
Назаренко Дмитро Миколайович	0	0		
	✗	✗		
Прокоп'єв Андрій Костянтинович	0	2		
	✗	✓		
Проява Владислав Вікторович	0	2		
	✗	✓		
Слобожанін Ілля Іванович	0	2		
	✗	✓		
Сперисенко Олександр Валерійович	0	2		
	✗	✓		
Степашкін Роман Романович	4	2		
	✓	✓		
Таха Загер Валерійович	4	2		
	✓	✓		

Рис. 3.19 Виставлення оцінок та пропусків викладачем.

Кількість стовпчиків пар формується виходячи з кількості пар за семестр, біля кожного студента можна поставити оцінку та пропуск. Також останні стовпчики дають інформацію про результат екзаменаційної сесії, середній бал за пару, загальний бал та кількість пропусків (рис. 3.20):

18	19	20	Екз. сесія	Середній бал	Загальний бал	Кількість пропусків
Оцінка/відвідування						
				4,5	9	0
				3,5	7	0
				3,5	7	0
				2,5	5	0
				0	0	2
				2,5	5	1
				0	0	2
				2	4	1
				4	8	0
				3	6	0
				2	4	0
				2	4	0
				2	4	0
				3,5	7	0
				0	0	2
				4	8	0
				0	0	2
				1	2	1
				1	2	1
				1	2	1
				1	2	1
				3	6	0
				3	6	0

Рис. 3.20 Виставлення оцінок та пропусків викладачем (продовження таблиці).

Як видно, доки не будуть проставлені бали за останню пару, поле "Екзаменаційна сесія" буде не активним. Також є стовпчик з середнім балом по усім парам, загальний бал і кількість пропусків.

В рамках дипломного проекту також розглянемо формування звіту "Самоаналіз". Це таблиця з загальною успішністю студентів за дисциплінами, де вираховується абсолютна та якісна успішність (рис. 3.21).

Назва дисциплін (П.І.Б. викладача)	Факультет	Курс	Група	Підсумковий контроль:																								
				Контингент студентів (осіб)	Чисельність студентів, які допущені до підсумкового модульного контролю (осіб)	Склали підсумковий модульний контроль (осіб):							Результати з урахуванням заліку (осіб):						Результати з урахуванням іспиту (осіб):						Якість навчання:		
						90-100	82-89	75-81	69-74	60-68	одержали незадовіль-ну оцінку:		90-100	82-89	75-81	69-74	60-68	одержали незадовіль-ну оцінку:		90-100	82-89	75-81	69-74	60-68	одержали незадовіль-ну оцінку:		Абсолютна успішність	Якісна успішність
											35-59	0-34						35-59	0-34						35-59	0-34		
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Захист комп'ютерних систем і мереж	ФОАІС	4	11	17	17	5	0	2	1	7	0	2	6	0	7	0	4	0	0	0	0	0	0	0	0	0	100,0%	76,5%
Менеджмент проектів програмного забезпечення	ФОАІС	4	7	27	27	13	0	5	0	0	6	3	13	0	5	0	9	0	0	0	0	0	0	0	0	0	100,0%	66,7%
Менеджмент проектів програмного забезпечення	ФОАІС	4	10	18	17	5	0	4	2	1	3	3	5	0	6	0	6	0	0	0	0	0	0	0	0	0	94,4%	61,1%
Управління проектами інформатизації	ФОАІС	4	2	26	25	9	4	2	5	2	0	4	9	4	3	6	0	3	0	0	0	0	0	0	0	0	84,6%	61,5%
Управління проектами інформатизації	ФОАІС	4	1	21	21	7	3	0	1	0	1	9	7	3	1	0	1	8	1	0	0	0	0	0	0	0	57,1%	52,4%
Безпека інформаційних систем та мереж	ФОАІС	1	1м	21	21	3	2	0	1	5	0	10	0	0	0	0	0	0	0	13	0	0	3	5	0	0	100,0%	61,9%
Безпека програм та даних	ФОАІС	4	7	27	27	8	1	7	0	4	2	5	0	0	0	0	0	0	0	12	2	4	2	7	0	0	100,0%	66,7%
Безпека програм та даних	ФОАІС	4	10	18	18	4	2	6	0	5	0	1	0	0	0	0	0	0	0	8	1	1	1	7	0	0	100,0%	55,6%
Алгоритми та структури даних	ФОАІС	3	6	28	27	3	2	2	1	8	6	6	0	0	0	0	0	0	0	4	1	1	4	11	5	1	75,0%	21,4%
Методи і засоби передачі даних	ФОАІС	3	6	28	25	9	1	1	3	1	6	7	0	0	0	0	0	0	0	7	3	4	0	5	5	1	67,9%	50,0%
Методи і засоби передачі даних	ФОАІС	3	7	20	20	1	1	4	5	6	3	0	0	0	0	0	0	0	0	1	3	3	5	8	0	0	100,0%	35,0%
Економіка і організація інформаційного бізнесу	ФОАІС	3	7	20	20	4	2	5	1	1	2	5	0	0	0	0	0	0	0	3	3	7	0	2	5	0	75,0%	65,0%
Економіка і організація інформаційного бізнесу	ФОАІС	3	6	28	26	5	1	1	1	4	2	14	0	0	0	0	0	0	0	5	2	5	0	14	0	0	92,9%	42,9%
Архітектура та проектування програмного забезпечення	ФОАІС	3	7	20	20	3	8	3	5	1	0	0	0	0	0	0	0	0	0	3	9	4	3	1	0	0	100,0%	80,0%

Рис. 3.21 Звіт "Самоаналіз".

3.2 Розробка блокчейну

Блокчейн зможе бачити тільки super-user. Маємо наступні вимоги:

- перегляд ланцюжка блоків
- перегляд тіла блоку з транзакціями
- interface Transaction, який реалізують декілька класів

Interface Transaction виглядає наступним чином (рис. 3.22):

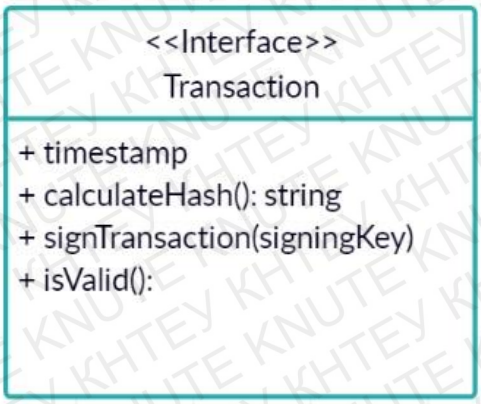

Рис. 3.22 Interface Transaction.

Ми вносимо різні дані у блокчейн, будь то новий викладач, чи внесення оцінки, тому необхідні різні класи транзакцій, які реалізують інтерфейс Transaction, в залежності від типу транзакції. Приклад для класу CreateStudentTransaction (рис. 3.23):

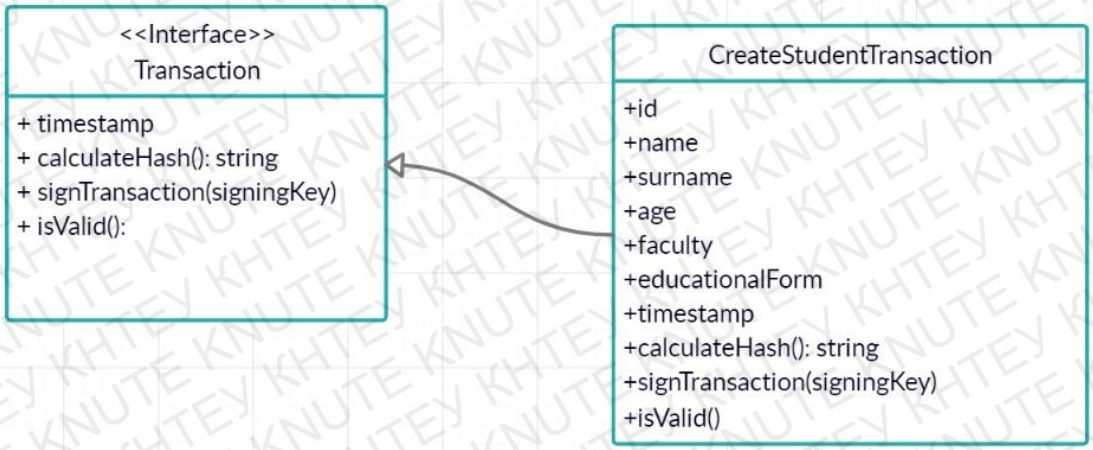

Рис. 3.23 Приклад класу на створення транзакції студента.

Кожен клас транзакції просто додає свої необхідні поля. Визначивши необхідні класи ми створили новий компонент "transaction-table" для ланцюга транзакцій та "block-view" для представлення вмісту інформації у блоку. Ланцюг блоків виглядає наступним чином и завжди починається з порожнього Genesis-блоку (рис. 3.24):

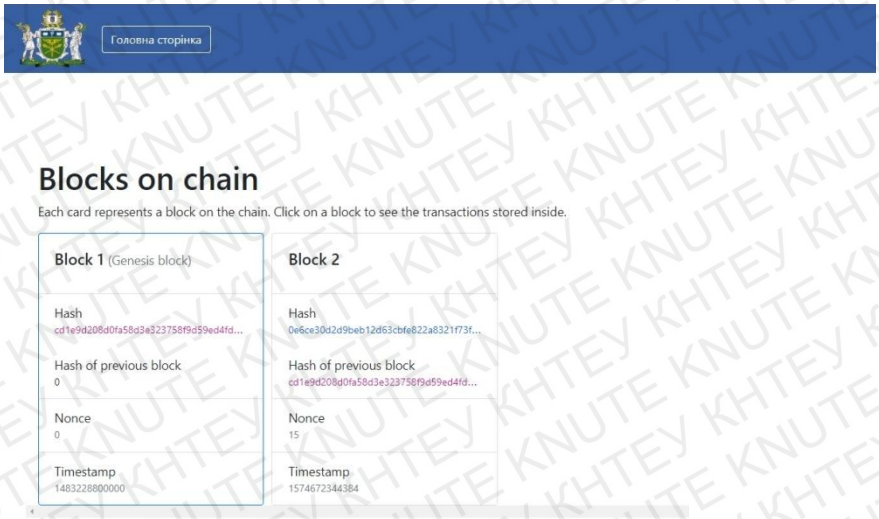

Рис. 3.24 Ланцюг блоків.

Перейдемо у вкладку "Створення нового студента" в кабінеті адміністратора. Додамо нового студента з наступними даними: id:0, name: "Дмитро", surname: "Кубишев", age: 23, faculty: "ФОАІС", educationalForm: "денна". Новий блок буде містити транзакцію по створенню студента і містити усю інформацію, що ми занесли (рис. 3.25):

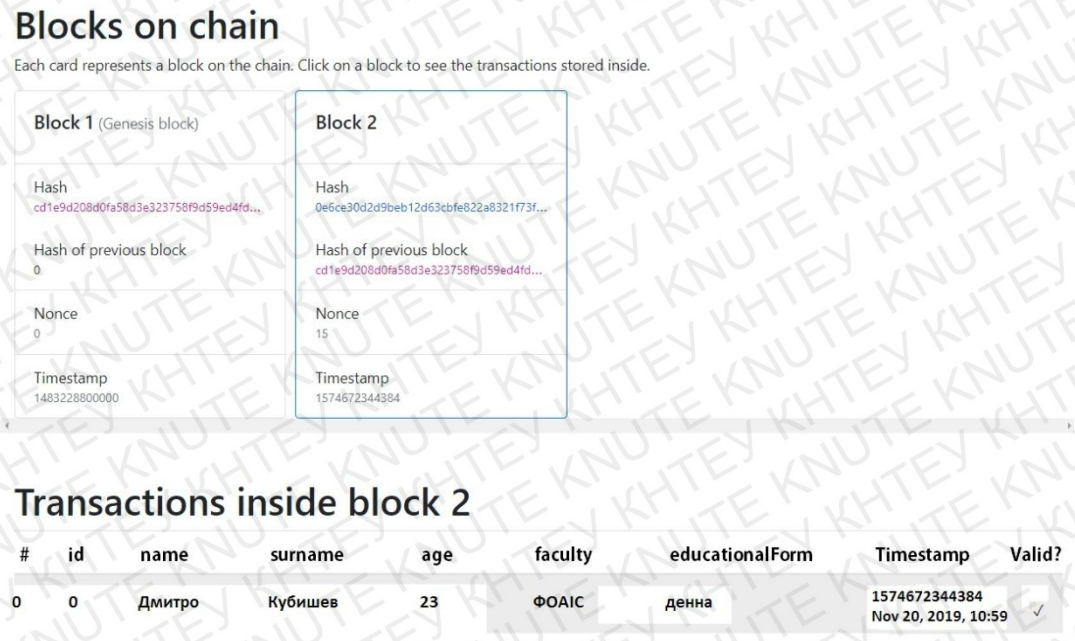

Рис. 3.25 Тіло блока з транзакцією на створення нового студента.

3.3 Висновки до розділу 3.

В результаті роботи над цим розділом було описано наступні кроки, що знадобилися для розробки веб-сервісу:

- інсталяція Node.js;
- створення нового проекту Angular;
- визначено файлову структуру проекту, що дуже важливо для зрозумілої навігації;
- розробка маршрутизації веб-додатку;
- описано принципи Material Design та підключено Angular-бібліотеку;
- система авторизації через Google аккаунт за допомогою Google Firebase;
- верстування кабінетів користувачів;
- опрацювання даних, що занесені в блокчейн;
- робота з даними різними звітами ("Самоаналіз");
- відображення ланцюжка-блоків у режимі адміністратора;
- відображення вмісту (тіла) блоків з різними типами транзакцій.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

У ході виконання дипломної роботи були більш докладно вивчені і викладені наступні питання:

- аналітичний огляд предметної області;
- написання технічного завдання програмного продукта;
- розробка блокчейну та ланцюга блоків;
- розробка веб-сервісу, яким можуть користуватися як викладачі, так і студенти;
- обгрунтовано вагомі переваги блокчейну над централізованими серверами;
- робота з Angular Framework.

Під час роботи було створено веб-сервіс, за допомогою якого користувачі можуть поліпшити якість освітнього процесу та систематизувати свою поточну успішність. Викладачі матимуть змогу вести веб-журнал, у якому будуть внесені усі оцінки та пропуски студентів. Програмний продукт, що був створений у ході виконання дипломної роботи, хоч і є прототипом, наглядно демонструє переваги та захисні механізми технології блокчейн та швидкість розробки за допомогою Angular.JS. Оскільки ланцюг блоків є майже стовідсотковим захистов від будь-яких уражень його подальше використання згодом дійде усіх сфер життя, у тому числі і освіти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. “What is Ethereum? A Step-by-Step Beginners Guide” [Електронний ресурс].- Режим доступу: https://blockgeeks.com/guides /ethereum
2. «Гідро Рейндроп відкрита аутентифікація на блокчейні» [Електронний ресурс]. - Режим доступу: https://www. Hydrogenplat form.com/white-papers/ Hydro_Raindrop_White_Paper_Ukranian
3. Narayanan A. Bitcoin and Cryptocurrency Technologies / A. Narayanan, J. Bonneau, E. Felten, A. Miller, S. Goldfeder. – Princeton: Princeton University Press, 2016. – 308 p.
4. TypeScript. The Definitive Guide [Електронний ресурс]. — Режим доступу: https://basarat.gitbooks.io/typescript/content/docs/getting-started. html
5. Архірейська Н. В. Блокчейн – інноваційна технологія постіндустріальної економіки [Електронний ресурс] / Н. В. Архірейська // Бізнес Інформ. - 2017. - № 7. - Режим доступу: http://nbuv.gov.ua/UJRN/binf_2017_7_21
6. Вікторов В. В. Розподілені обчислення як інструмент блокчейн технологій [Електронний ресурс] / В. В. Вікторов // Проблеми економіки та політичної економії. - 2019. - № 1. - Режим доступу: http://nbuv.gov.ua/UJRN/pepe_2019_1_13
7. Волошин В. С. Ожидаемые и реальные риски в блокчейн-технологиях [Електронний ресурс] / В. С. Волошин // Теоретичні і практичні аспекти економіки та інтелектуальної власності. - 2018. - Вип. 17. - Режим доступу: http://nbuv.gov.ua/UJRN/Tpaeiv_2018_17_3
8. Глибовець М. М. Методологічні аспекти розробки платформи для залучення інвестицій в освіту на базі інструменту ICO (Initial Coin Offering) технології блокчейн – стартап НаУКМА "Knowledge Measurable Assets" / М. М. Глибовець, Є. І. Невмержицький, К. С.

Гороховський, Є. Й. Грабов // Наукові записки НаУКМА. Комп'ютерні науки. - 2017. - Т. 198. - С. 47-53.

9. Давидова І. В. Технологія блокчейн: перспективи розвитку в Україні [Електронний ресурс] / І. В. Давидова // Часопис цивілістики. - 2017. - Вип. 26. - Режим доступу: http://nbuv.gov.ua/UJRN/Chac_2017_26_10

10. Діордієв В. О. Засади створення фінансового інноваційного хабу в Україні / В. О. Діордієв [Електронний ресурс]. – Режим доступу: http://www.m.nayka.com.ua/?op=1&j =efektyvna-ekonomika&s=ua&z=5207

11. Документація мови TypeScript [Електронний ресурс]. — Режим доступу: https://www.typescriptlang.org/docs/tutorial.html

12. Дудник А. С. Анализ технологии применения блокчейн совместно с технологией интернет вещей для обработки и хранения результатов измерений [Електронний ресурс] / А. С. Дудник, О. Г. Чолышкина, М. Г. Луцкий // Молодий вчений. - 2018. - № 5(1). - Режим доступу: http://nbuv.gov.ua/UJRN/molv_

13. Кудін А. М. Технологія блокчейн: питання аналізу та синтезу [Електронний ресурс] / А. М. Кудін, Б. А. Коваленко, І. В. Швідченко // Кибернетика и системный анализ. - 2019. - Т. 55, № 3. - С. 164-172

14. Липницкий Д. В. Возможности и вызовы для блокчейн в новой индустриализации [Електронний ресурс] / Д. В. Липницкий. - Режим доступу: http://nbuv.gov.ua/UJRN/econpr_2019_1_7

15. Миронець І. В. Технологія блокчейн: аналіз загроз для блокчейн-систем із механізмом досягнення консенсусу [Електронний ресурс] / І. В. Миронець, А. В. Шкребтій, В. А. Борисенко . - Режим доступу: http://nbuv.gov.ua/UJRN/sntuts_2018_29_2_31

16. Павел Козловский, Питер Бэкон Дарвин Разработка веб-приложений с использованием AngularJS. – Пер. с англ. Киселева А. Н., М.: ДМК Пресс, 2014. – 394с.

17. Сергеєв Є. І. Порівняння мов програмування Typescript та Javascript в розробці сучасних веб-додатків [Електронний ресурс] / Є. І. Сергеєв, А. П. Прасолов // International scientific journal. - 2016. - № 7. - Режим доступу: http://nbuv.gov.ua/UJRN/mnj_2016_7_28

18. Спасітєлєва С. О. Перспективи розвитку додатків блокчейн в Україні [Електронний ресурс] / С. О. Спасітєлєва, В. Л. Бурячок // Кібербезпека: освіта, наука, техніка. - 2018. - № 1. - Режим доступу: http://nbuv.gov.ua/UJRN/cest_2018_1_6

19. Файн Я., Моисеев А. Angular и TypeScript. Сайтостроение для профессионалов. — СПб.: Питер, 2018. — 464 с.

20. Фримен А. Ф88 Angular для профессионалов. — СПб.: Питер, 2018. — 800 с

21. Чернишов Д. У майбутньому Україна переведе всю цифрову державну інформацію на блокчейн-платформу [Електронний ресурс]. – Режим доступу: http://www.pravove-pole.info/novini/u-majbutnomu-ukraina-perevede-vsju-cyfrovu-derzhavnu-informaciju-na-blokchejnplatformu-denys-chernyshov/

ДОДАТКИ

Додаток А (блокчейн)

```
const crypto = require('crypto');
const EC = require('elliptic').ec;
const ec = new EC('secp256k1');
const debug = require('debug')('savjeecoin:blockchain');

interface Transaction {
 /**
  * @param timestamp;
  */
 constructor() {
 }

 /**
  * Creates a SHA256 hash of the transaction
  *
  * @returns {string}
  */
 calculateHash() {
 }
 signTransaction(signingKey) {
 }
 }
 isValid() {
 }
}

class CreateStudentTransaction {
 /**
  * @param {number} id

  * @param {number} name

  * @param {number} surname

  * @param {number} age

  * @param {number} faculty

* @param {number} educationalForm

  * @param {string} fromAddress;

   * @param {string} toAddress;
  */
```

```
  constructor(id, name, surname, age, faculty, educationalForm) {
    this.id = id;
    this.name = name;
    this.surname = surname;
    this.age = age;
    this.faculty = faculty;
    this.educationalForm = educationalForm;
    this.timestamp = Date.now();
  }

  /**
   * Creates a SHA256 hash of the transaction
   *
   * @returns {string}
   */
  calculateHash() {
    return crypto.createHash('sha256').update(this.id + this.name + this.surname + this.age +
this.faculty + this.educationalForm + this.timestamp).digest('hex');
  }

  /**
   * Signs a transaction with the given signingKey (which is an Elliptic keypair
   * object that contains a private key). The signature is then stored inside the
   * transaction object and later stored on the blockchain.
   *
   * @param {string} signingKey
   */
  signTransaction(signingKey) {

    if (signingKey.getPublic('hex') !== this.fromAddress) {
      throw new Error('You cannot sign transactions for other wallets!');
    }

    // Calculate the hash of this transaction, sign it with the key
    // and store it inside the transaction obect
    const hashTx = this.calculateHash();
    const sig = signingKey.sign(hashTx, 'base64');

    this.signature = sig.toDER('hex');
  }
```

```
  /**
   * Checks if the signature is valid (transaction has not been tampered with).
   * It uses the fromAddress as the public key.
   *
   * @returns {boolean}
   */
  isValid() {
    // If the transaction doesn't have a from address we assume it's a
    // mining reward and that it's valid. You could verify this in a
    // different way (special field for instance)
    if (this.fromAddress === null) return true;

    if (!this.signature || this.signature.length === 0) {
      throw new Error('No signature in this transaction');
    }

    const publicKey = ec.keyFromPublic(this.fromAddress, 'hex');
    return publicKey.verify(this.calculateHash(), this.signature);
  }
}
class Block {
  /**
   * @param {number} timestamp
   * @param {Transaction[]} transactions
   * @param {string} previousHash
   */

  constructor(timestamp, transactions, previousHash = '') {
    this.previousHash = previousHash;
    this.timestamp = timestamp;
    this.transactions = transactions;
    this.nonce = 0;
    this.hash = this.calculateHash();
  }

  /**
   * Returns the SHA256 of this block (by processing all the data stored
   * inside this block)
   *
   * @returns {string}
   */
```

```
 calculateHash() {
   return crypto.createHash('sha256').update(this.previousHash + this.timestamp +
JSON.stringify(this.transactions) + this.nonce).digest('hex');
  }

  /**
   * Starts the mining process on the block. It changes the 'nonce' until the hash
   * of the block starts with enough zeros (= difficulty)
   *
   * @param {number} difficulty
   */
  mineBlock(difficulty) {
   while (this.hash.substring(0, difficulty) !== Array(difficulty + 1).join('0')) {
    this.nonce++;
    this.hash = this.calculateHash();
   }

   debug(`Block mined: ${this.hash}`);
  }
  /**
   * Validates all the transactions inside this block (signature + hash) and
   * returns true if everything checks out. False if the block is invalid.
   *
   * @returns {boolean}
   */
  hasValidTransactions() {
   for (const tx of this.transactions) {
    if (!tx.isValid()) {
     return false;
    }
   }

   return true;
  }
}
class Blockchain {
  constructor() {
   this.chain = [this.createGenesisBlock()];
   this.difficulty = 2;
   this.pendingTransactions = [];
   this.miningReward = 100;
  }
```

```
createGenesisBlock() {
  return new Block(Date.parse('2017-01-01'), [], '0');
 }

 /**
  * Returns the latest block on our chain. Useful when you want to create a
  * new Block and you need the hash of the previous Block.
  *

  * @returns {Block[]}
  */
 getLatestBlock() {
  return this.chain[this.chain.length - 1];
 }

 /**
  * Takes all the pending transactions, puts them in a Block and starts the
  * mining process. It also adds a transaction to send the mining reward to
  * the given address.
  *
  * @param {string} miningRewardAddress
  */
 minePendingTransactions(miningRewardAddress) {
  const rewardTx = new Transaction(null, miningRewardAddress, this.miningReward);
  this.pendingTransactions.push(rewardTx);
  const block = new Block(Date.now(), this.pendingTransactions, this.getLatestBlock().hash);
  block.mineBlock(this.difficulty);

  debug('Block successfully mined!');
  this.chain.push(block);

  this.pendingTransactions = [];
 }

 /**
  * Add a new transaction to the list of pending transactions (to be added
  * next time the mining process starts). This verifies that the given
  * transaction is properly signed.
  *
  * @param {Transaction} transaction
  */
 addTransaction(transaction) {
```

```
  if (!transaction.fromAddress || !transaction.toAddress) {
    throw new Error('Transaction must include from and to address');
  }

  // Verify the transactiion
  if (!transaction.isValid()) {
    throw new Error('Cannot add invalid transaction to chain');
  }

  if (transaction.amount <= 0) {
    throw new Error('Transaction amount should be higher than 0');
  }

  // Making sure that the amount sent is not greater than existing balance
  if (this.getBalanceOfAddress(transaction.fromAddress) < transaction.amount) {
    throw new Error('Not enough balance');
  }

  this.pendingTransactions.push(transaction);
  debug('transaction added: %s', transaction);
}

/**
 * Returns the balance of a given wallet address.
 *
 * @param {string} address
 * @returns {number} The balance of the wallet
 */
getBalanceOfAddress(address) {
  let balance = 0;

  for (const block of this.chain) {
    for (const trans of block.transactions) {
      if (trans.fromAddress === address) {
        balance -= trans.amount;
      }

      if (trans.toAddress === address) {
        balance += trans.amount;
      }
    }
  }
```

```
  debug('getBalanceOfAdrees: %s', balance);
  return balance;
}

/**
 * Returns a list of all transactions that happened
 * to and from the given wallet address.
 *
 * @param  {string} address
 * @return {Transaction[]}
 */
getAllTransactionsForWallet(address) {
  const txs = [];

  for (const block of this.chain) {
    for (const tx of block.transactions) {
      if (tx.fromAddress === address || tx.toAddress === address) {
        txs.push(tx);
      }
    }
  }

  debug('get transactions for wallet count: %s', txs.length);
  return txs;
}

/**
 * Loops over all the blocks in the chain and verify if they are properly
 * linked together and nobody has tampered with the hashes. By checking
 * the blocks it also verifies the (signed) transactions inside of them.
 *
 * @returns {boolean}

 */
isChainValid() {
  // Check if the Genesis block hasn't been tampered with by comparing
  // the output of createGenesisBlock with the first block on our chain
  const realGenesis = JSON.stringify(this.createGenesisBlock());

  if (realGenesis !== JSON.stringify(this.chain[0])) {
    return false;
  }
```

```
for (let i = 1; i < this.chain.length; i++) {
    const currentBlock = this.chain[i];

    if (!currentBlock.hasValidTransactions()) {
      return false;
    }

    if (currentBlock.hash !== currentBlock.calculateHash()) {
      return false;
    }
   }

   return true;
  }
}
```

Додаток Б (Router)

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import {Routes, RouterModule} from '@angular/router';

import { AppComponent } from './app.component';
import { MainComponent } from './pages/main/main.component';
import { CreateStudentComponent } from './pages/admin/create-student/create-
student.component';
import {FormsModule} from '@angular/forms';
import { CreateGroupComponent } from './pages/admin/create-group/create-group.component';
import { CreateLecturerComponent } from './pages/admin/create-lecturer/create-
lecturer.component';
import { CreateSubjectComponent } from './pages/admin/create-subject/create-
subject.component';
import { StudentMarksComponent } from './pages/student/student-marks/student-
marks.component';
import { StudentSkipsComponent } from './pages/student/student-skips/student-
skips.component';
import { StudentCardComponent } from './pages/student/student-card/student-card.component';
import { StudentStatisticsComponent } from './pages/student/student-statistics/student-
statistics.component';
import { LecturerSetMarkComponent } from './pages/lecturer/lecturer-set-mark/lecturer-set-
mark.component';
import { LecturerSetSkipComponent } from './pages/lecturer/lecturer-set-skip/lecturer-set-
skip.component';
```

```
import { LecturerCardComponent } from './pages/lecturer/lecturer-card/lecturer-
card.component';
import { LecturerJournalComponent } from './pages/lecturer/lecturer-journal/lecturer-
journal.component';
import { LecturerExaminationComponent } from './pages/lecturer/lecturer-examination/lecturer-
examination.component';

const routes: Routes = [
  {path: '', component: MainComponent},
  {path: 'create-student', component: CreateStudentComponent},
  {path: 'create-subject', component: CreateSubjectComponent},
  {path: 'create-lecturer', component: CreateLecturerComponent},
  {path: 'create-group', component: CreateGroupComponent},
  {path: 'lecturer-journal', component: LecturerJournalComponent},
  {path: 'lecturer-set-mark', component: LecturerSetMarkComponent},
  {path: 'lecturer-set-skip', component: LecturerSetSkipComponent}
]

@NgModule({
  declarations: [
    AppComponent,
    MainComponent,
    CreateStudentComponent,
    CreateGroupComponent,
    CreateLecturerComponent,
    CreateSubjectComponent,
    StudentMarksComponent,
    StudentSkipsComponent,
    StudentCardComponent,
    StudentStatisticsComponent,
    LecturerSetMarkComponent,
    LecturerSetSkipComponent,
    LecturerCardComponent,
  ],
  imports: [
    BrowserModule,
    RouterModule.forRoot(routes),
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Додаток В (Frontend таблиця блоків)

```
<div class="card" [class.border-primary]="isSelectedBlock()">
  <div class="card-body">
   <h5 class="card-title">Block {{ getBlockNumber() }} <small class="text-muted"
*ngIf="block.previousHash == 0">(Genesis block)</small></h5>
  </div>

   <ul class="list-group list-group-flush">
   <li class="list-group-item">
    <span class="">Hash</span>
    <br>
    <div class="text-truncate" [style.color]="'#' + block.hash.substring(0,6)">
     <small>{{ block.hash }}</small>
    </div>

  <br>
  <span class="">Hash of previous block</span>
    <br>

    <div class="text-truncate" [style.color]="'#' + block.previousHash.substring(0,6)">
     <small>{{ block.previousHash }}</small>
    </div>
   </li>

   <li class="list-group-item">
    <span class="">Nonce</span><br>
    <div class="text-truncate text-muted">
     <small>{{ block.nonce }}</small>
    </div>
   </li>

   <li class="list-group-item">
    <span class="">Timestamp</span><br>
    <div class="text-truncate text-muted">
     <small>{{ block.timestamp }}</small>
    </div>
   </li>
  </ul>
 </div>
```