МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ АЕРОНАВІГАЦІЇ, ЕЛЕКТРОНІКИ ТА ТЕЛЕКОМУНІКАЦІЙ
КАФЕДРА ЕЛЕКТРОНІКИ, РОБОТОТЕХНІКИ І ТЕХНОЛОГІЙ
МОНІТОРИНГУ ТА ІНТЕРНЕТУ РЕЧЕЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Шутко В.М.
«____»_____ 2021 р.

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА
ЗІ СПЕЦІАЛЬНОСТІ 171 «ЕЛЕКТРОНІКА»
ОПП «ЕЛЕКТРОННІ СИСТЕМИ»

Тема: «Система управління інтернет банком»

Виконавець
студент групи ЕС-413Б _____ Дячук Андрій Володимирович
Керівник
д.т.н., професор _____Шутко Володимир Миколайович

Нормоконтролер _____ Сініцин Р.Б.

КИЇВ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут аеронавігації електроніки та телекомунікацій

Кафедра електроніки, робототехніки і технологій моніторингу та інтернету речей

Напрям (спеціальність) 171 «Електроніка»

(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Шутко В.М.

«_____»_____2021р.

ЗАВДАННЯ

на виконання дипломної роботи

Дячука Андрія Володимировича

(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи: «Система управління інтернет банком» затверджена наказом ректора від «__1__»__квітня_____2021_ р. №____526/ст_____

2. Термін виконання роботи : з __17травня 2021р по _18 червня 2021р._

3. Вихідні дані до роботи : Система управління банком веде щоденні підрахунки підрахунків як повне банківське обслуговування. Він може вести записи про тип рахунку, форму відкриття рахунку, депозит, зняття коштів, звіт про транзакції, форму відкриття рахунків. Захоплюючою частиною цього проекту є те, що він відображає звіт про транзакції.

4. Зміст пояснювальної записки: Розділ 1 : Вступ..Аналіз системи , створення діаграм ERD , DFD ; Розділ 2 : Дизайн модуля та проектування бази данних ;

Розділ 3 : Тестування системи. План тесту;

5. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Підбір літератури	04.04.21	виконав
2	Настройка програмного середовища	05.04.21	виконав
3	Розробка діаграм ERD і DFD	15.04.21-23.04.21	виконав
4	Верстка сайта	09.05.21-10.05.21	виконав
5	Розробка модуля Адмін і Користувач	11.05.21-12.05.21	виконав
6	Розробка бази даних і підключення драйвера за допомогою JDBC	13.05.21-14.05.21	виконав
7	Виправлення ошибок	15.05.21-17.05.21	виконав
8	Тестування програми	18.05.21-20.05.21	виконав
9	Написання висновка	21.05.21-23.05.21	виконав
10	Оформлення роботи	27.05.21	виконав

6. Дата видачі завдання: “__03___"___червня_____ 2021_ р.

Керівник дипломної роботи (проекту) _____Шутко В.М.

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____Дячук А.В.____

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Основна мета створення веб-додатки на Java з використанням таких технологій як Java, JSP, MySql, Hibernate , сервер - apache-tomcat-8.5.13. Ця система допоможе вести записи про банківські рахунки, вести записи транзакцій клієнтів. Легко відслідковувати всю банківську діяльність централізована система для підтримки всієї банківської діяльності.

У додатку є два основних учасника:

Адмін (банк)

Користувач (Замовник)

ЗМІСТ

РОЗДІЛ 1

Вступ

1.1. Аналіз системи

Домен “Система управління банком” веде щоденні підрахунки як повне банківське обслуговування. Він може вести записи про тип рахунку, форму відкриття рахунку, депозит, зняття коштів, звіт про транзакції, форму відкриття рахунків. Цікавою частиною цього проекту є те, що він відображає звіт про трансакції.

СИНОПСИС

Домен “Система управління банком” веде щоденні підрахунки підрахунків як повне банківське обслуговування. Він може вести записи про тип рахунку, форму відкриття рахунку, депозит, зняття коштів, звіт про транзакції, форму відкриття рахунків. Захоплюючою частиною цього проекту є те, що він відображає звіт про транзакції.

АДМІНІСТРАТИВНИЙ МОДУЛЬ

Цей модуль є основним модулем, який виконує всі основні операції в системі. Основними операціями в системі є:

- Додати рахунок
- Внести гроші
- Вивести гроші
- Тип рахунку
- Виконайте транзакцію

• Перегляньте транзакцію

СИСТЕМНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ

АНАЛІЗ СИСТЕМИ

Системний аналіз - це процес збору та інтерпретації фактів, діагностування проблем та інформації, щоб рекомендувати вдосконалення системи. Це діяльність з вирішення проблем, яка вимагає інтенсивного спілкування між користувачами системи та розробниками системи. Системний аналіз або дослідження є важливою фазою будь-якого процесу розробки системи. Система вивчається до найдрібніших деталей та аналізується. Системний аналітик виконує роль допитувача і глибоко зупиняється на роботі нинішньої системи. Система розглядається як єдине ціле та визначаються вхідні дані в систему. Результати діяльності організацій відстежуються на різних процесах.Системний аналіз стосується ознайомлення з проблемою, виявлення відповідних змінних та змінних, аналізу та синтезу різних факторів та визначення оптимального чи принаймні задовільного рішення чи програми дій.

Детальне вивчення процесу необхідно робити за допомогою різних методів, таких як співбесіди, анкети тощо. Дані, зібрані цими джерелами, повинні бути детально вивчені, щоб дійти висновку. Висновок - розуміння того, як функціонує система. Ця система називається існуючою. Зараз існуюча система ретельно вивчається та визначаються проблемні області. Зараз конструктор функціонує як вирішувач проблем і намагається розібратися з труднощами, з якими стикається підприємство. Рішення подаються як пропозиції. Потім пропозиція аналізується з існуючою системою аналітично і вибирається найкраща. Пропозиція надається користувачеві для схвалення

користувачем. Пропозиція розглядається на запит користувача та вносяться відповідні зміни. Цей цикл закінчується, як тільки користувач задовольняється пропозицією.

Попереднє дослідження - це процес збору та інтерпретації фактів з використанням інформації для подальших досліджень системи. Попереднє дослідження - це вирішення проблем, що вимагає інтенсивного спілкування між користувачами системи та розробниками системи. Він проводить різні техніко-економічні обґрунтування. У цих дослідженнях можна отримати приблизну цифру системної діяльності, з якої можна прийняти рішення щодо стратегій, яких слід дотримуватися для ефективного системного вивчення та аналізу.

ІСНУЮЧА СИСТЕМА

У існуючій системі транзакції здійснюються лише вручну, але в запропонованій системі ми повинні обчислити всі банківські операції за допомогою програмного забезпечення Банківська система.

ПРОБЛЕМИ З ІСНУЮЧОЮ СИСТЕМОЮ

- Недостатня безпека даних.
- Більша робоча сила.
- Затрата часу.
- Споживання великої кількості попередньої роботи.
- Потрібні ручні розрахунки.
- Немає прямої ролі для вищих посадових осіб.
- Пошкодження машин через відсутність уваги.

Щоб уникнути всіх цих обмежень та зробити роботу точнішою, потрібно комп’ютеризувати систему.

ПРОПОЗИЦІЙНА СИСТЕМА

Метою запропонованої системи є: розробити систему вдосконалених приміщень. Запропонована система може подолати всі обмеження існуючої системи. Система забезпечує належний захист та зменшує ручну роботу.

ПЕРЕВАГИ ПРОПОЗИЦІЙНОЇ СИСТЕМИ

Система дуже проста в розробці та реалізації. Система вимагає дуже низьких системних ресурсів, і система буде працювати майже у всіх конфігураціях. Він має такі функції:

- Захист даних.
- Забезпечення точності даних.
- Належний контроль вищих посадових осіб.
- Зменшення пошкодження машин.
- Звести до мінімуму введення даних вручну.
- Мінімальний час, необхідний для різноманітної обробки.
- Більша ефективність.
- Краще обслуговування.
- Зручність та інтерактивність.
- Мінімальний час.

ТЕХНІЧНА ДОСЛІДЖЕННЯ

Проводиться техніко-економічне обґрунтування, щоб побачити, чи буде проект, який завершується, відповідати меті організації за обсягом роботи, зусиль та часу, витраченого на нього. Техніко-економічне обґрунтування дозволяє розробнику передбачити майбутнє проекту та корисність. Техніко-

економічне обґрунтування пропозиції системи здійснюється відповідно до її працездатності, яка полягає у впливі на організацію, здатності задовольнити потреби користувачів та ефективному використанні ресурсів. Таким чином, коли пропонується нова заявка, вона, як правило, проходить техніко-економічне обгрунтування до того, як її затвердять для розробки.

Документ надає можливість реалізації проекту, що розробляється, та перелічує різні сфери, які розглядались дуже ретельно під час техніко-економічного обґрунтування цього проекту, такого як технічні, економічні та експлуатаційні можливості. Нижче наводяться його особливості:

ТЕХНІЧНА ДОСТУПНІСТЬ

Спочатку систему слід оцінити з технічної точки зору. Оцінка цієї доцільності повинна базуватися на загальному проекті системної вимоги з точки зору введення, результату, програм та процедур. Визначивши структурну систему, розслідування повинно продовжувати, пропонуючи тип обладнання, необхідний метод розробки системи, для роботи системи після її проектування.

Технічні проблеми, порушені під час розслідування є:
Чи достатня існуюча технологія для запропонованої? Чи може система розширитися, якщо її розробити?

Проект слід розробляти таким чином, щоб необхідні функції та продуктивність були досягнуті в рамках обмежень. Проект розроблений в рамках новітніх технологій.Завдяки цій технології через деякий час може застаріти, оскільки жодна версія одного програмного забезпечення не підтримує старіші версії, система все одно може використовуватися. Отже,

цей проект має мінімальні обмеження. Система розроблена з використанням Java, проект технічно здійсненний для розробки.

ЕКОНОМІЧНА ДОСТУПНІСТЬ

Система, що розробляється, повинна бути обґрунтована витратами та вигодами. Критерії, що забезпечують зосередження зусиль на проекті, який дасть найкращі результати, повертаються не раніше якнайшвидше. Одним із факторів, що впливає на розробку нової системи, є витрати, які вона потребуватиме.

Нижче наведено деякі важливі фінансові питання, задані під час попереднього розслідування:

• Витрати проводять повне системне розслідування.

• Вартість апаратного та програмного забезпечення.

• Переваги у вигляді зменшення витрат або меншої кількості дорогих помилок.

Оскільки система розроблена в рамках проектної роботи, витрати на запропоновану систему не вимагають жодних ручних витрат. Також усі ресурси вже доступні, це вказує на те, що система економічно можлива для розвитку.

ПОВЕДІНСЬКА МОЖЛИВОСТЬ

Це включає такі запитання:

Чи достатня підтримка для користувачів?

Чи запропонована система завдасть шкоди?

Проект був би корисним, оскільки задовольняє цілі при розробці та встановленні. Усі поведінкові аспекти ретельно розглядаються і роблять висновок, що проект є поведінково здійсненним.

СИСТЕМНИЙ ДИЗАЙН

Дизайн - це перший крок на етапі розробки будь-якого розробленого продукту чи системи. Дизайн - це творчий процес. Хороший дизайн - запорука ефективної системи. Термін "проектування" визначається як "процес застосування різних методів і принципів з метою визначення процесу або системи з достатньою деталізацією, щоб дозволити її фізичну реалізацію". Це може бути визначено як процес застосування різних технік та принципів з метою визначення пристрою, процесу або системи з достатньою деталізацією, щоб дозволити його фізичну реалізацію. Розробка програмного забезпечення відповідає технічному ядру процесу розробки програмного забезпечення та застосовується незалежно від використовуваної парадигми розробки. Дизайн системи розробляє архітектурні деталі, необхідні для побудови системи або продукту. Як і у випадку з будь-яким систематичним підходом, це програмне забезпечення також пройшло найкращу з можливих стадій проектування, довівши всі рівні ефективності, продуктивності та точності. Етап проектування - це перехід від орієнтованого на користувача документа до документа до програмістів або персоналу баз даних. Проектування системи проходить дві фази розробки: логічне та фізичне проектування.

ЛОГІЧНИЙ ДИЗАЙН:

Логічний потік системи та визначення меж системи. Він включає наступні кроки:

• Переглядає поточну фізичну систему - її потоки даних, вміст файлів, обсяги, частоти тощо.

• Готує специфікації виводу - тобто визначає формат, вміст і частоту звітів.

• Готує специфікації введення - формат, вміст та більшість функцій введення.

• Готує специфікації редагування, безпеки та контролю.

• Вказує план впровадження.

• Готує логічний проектний потік інформаційного потоку, виведення, введення, контролю та плану реалізації. .

• Переглядає переваги, витрати, цільові терміни та системні обмеження.

ФІЗИЧНИЙ ДИЗАЙН:

Фізична система створює робочі системи, визначаючи специфікації конструкції, що повідомляють програмісти саме те, що повинна робити система кандидатів. Він включає наступні кроки.

• Спроектуйте фізичну систему.

• Вкажіть носії вводу та виводу.

• Спроектуйте базу даних та вкажіть процедури резервного копіювання.

• Спроектуйте фізичний потік інформації через систему та фізичний дизайн.

• Плануйте впровадження системи.

• Підготуйте графік переходів та цільову дату.

• Визначте навчання процедури, курси та графік.

• Розробіть план тестування та впровадження та вкажіть будь-яке нове апаратне / програмне забезпечення.

• Оновіть переваги, витрати, дату перетворення та системні обмеження

1.2. Діаграми ERD,DFD

Що таке діаграма ER?

ER Diagram - це спосіб розробити концептуальний дизайн бази даних. Концептуально це означає, якими будуть суті, які будуть відносини між цими сутностями, яку інформацію сутностей нам потрібно зберігати в базі даних.
Отже, щоб представити цю інформацію у вигляді діаграми, ми використовуємо діаграму ER, яка є абревіатурою від Entity Relationship Diagram, яка не що інше, як показує взаємозв'язок між сутностями (які є об'єктами) в системі.

Навіщо потрібна ER-діаграма в процесі розробки програми?
Коли ми створюємо будь-яка програма як розробник програмного забезпечення, ми повинні слідувати деякому циклу, тобто життєвого циклу розробки програмного забезпечення, який має кілька фаз, в якому до початку фактичного кодування одна фаза називається фазою проектування, де нам потрібно намалювати кілька діаграм, щоб чітке розуміння того, що ми збираємося реалізувати в додатку.
Отже, нам потрібно намалювати багато діаграм, одна з яких представляє собою діаграму ER, з якої можна зрозуміти, яку інформацію про об'єкти і відносинах ми будемо зберігати в базі даних. Нижче наведені причини, за якими потрібно ER:

- Ми отримали б чітке уявлення про конструкцію системи.
- З його допомогою ми можемо легко зіставити його з реляційної схемою.
- Це допоможе нам створити логічне уявлення про систему.

DFD-діаграми активно застосовуються при розробці програмного забезпечення. При цьому:
сховища даних - це електронні таблиці і бази даних,
зовнішні сутності - клієнти або інші бази даних, в тому числі, з інших програм (інтеграція і обмін даними),

процеси - це виконувані функції і модулі в системі.

Також DFD нотації зручні при аналізі, коли система розглядається з точки зору документообігу. При цьому можна наочно побачити, де зберігаються дані, яким чином проводиться обмін документацією, де в цьому процесі допущені помилки організації бізнес-процесів та ін. Але тут застосування DFD діаграм вимагає особливої обережності. Все ж це не опис бізнес-процесу як такого, а, скоріше, діаграма переміщення даних при реалізації бізнес-процесів. Але як допоміжний варіант, в тому числі, для наочної демонстрації клієнту існуючих проблем і методів оптимізації роботи, цей вид нотацій цілком підійде.

Наприклад, для виявлення проблем документообігу, дублювання документів або, навпаки, відсутньої документації або електронних даних в системі, дуже зручно створити окремо - опис бізнес-процесу, а потім до нього - DFD-нотацію. Або навпаки, попередньо для розуміння основ роботи бізнесу та особливостей реалізації документообігу створюється DFD-нотація. Вона допомагає виявити, наприклад, відсутність в системі автоматизації важливих документів, які насправді створюються (на папері), але в системі ніяк не відображаються. А потім вже будується оптимізований бізнес-процес з урахуванням виявлених нюансів документообігу.

DFD нотації - це просто!

Я вважаю, що DFD нотації - це дійсно багато простіше, ніж це здається на перший погляд. Головне, чітко розуміти обмеження побудови цього типу діаграм (відсутність умов, часу і т.д.) і застосовувати їх там, де саме такий підхід виявиться зручніше. Можливо, ви знайдете власні варіанти застосування DFD, які я вище не описати. У моєму переліку присутні тільки ті варіанти, які я використовую на практиці.

Що в DFD-нотаціях особливо зручно, тут не обов'язково дотримуватися строгих правил і синтаксису, як, наприклад, в BPMN. Ці нотації НЕ будуть здійснимими, вони потрібні для розуміння особливостей документообігу, структури та подальшої роботи з даними. А тому, якщо ваша діаграма зрозуміла і вам, і замовнику, якісь відступи від стандартів DFD цілком припустимі.

Малювати діаграми DFD можна, в принципі, де і як вам зручніше. Але якщо ви хочете працювати з декомпозицією, вибудовувати систему на різних рівнях деталізації, то «рисовалки» (Visio, Paint і тому подібні) доведеться забути. Вам будуть потрібні спеціалізовані програми для моделювання.

Особисто я користуюся програмою ERwin і всім її рекомендую. Одна з причин мого вибору - це особливості декомпозиції. У ERwin, як і в деяких інших подібних системах, існує можливість декомпозірованія DFD-процесів в форматі IDEF3, тобто основна діаграма буде в форматі DFD, і на найзагальнішому рівні ви будете бачити основні потоки даних і «вузли» їх обробки. А при декомпозиції ви зможете використовувати вже процесний підхід, що також буває дуже зручно для розробки великих систем або роботі з різними підрозділами бізнесу.

Діаграма ER

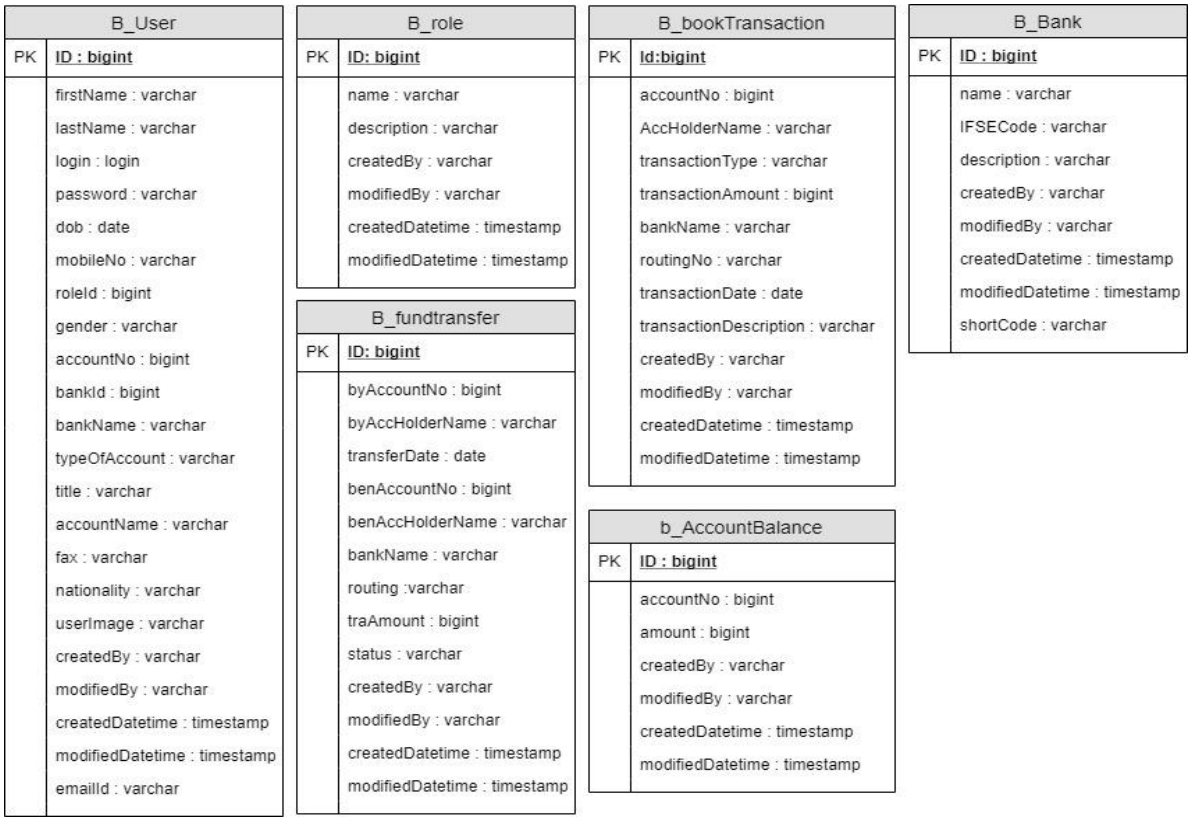


Рис 1.2.1

DFD Діаграма

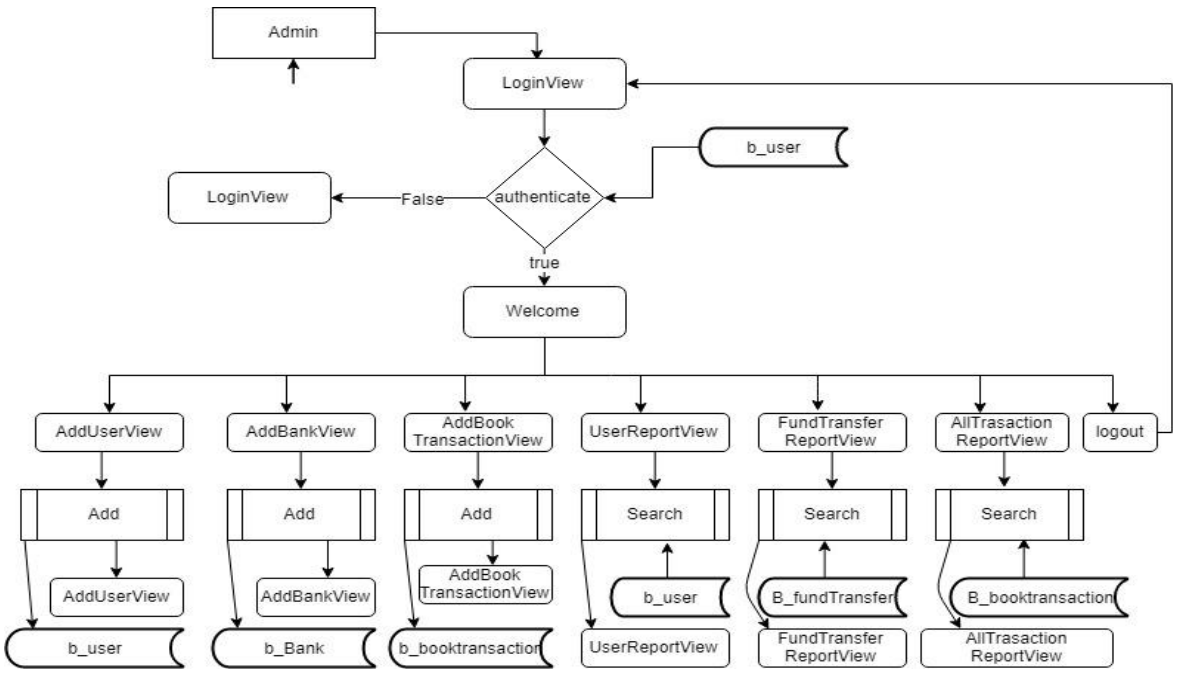


Рис 1.2.2

ER для менеджера

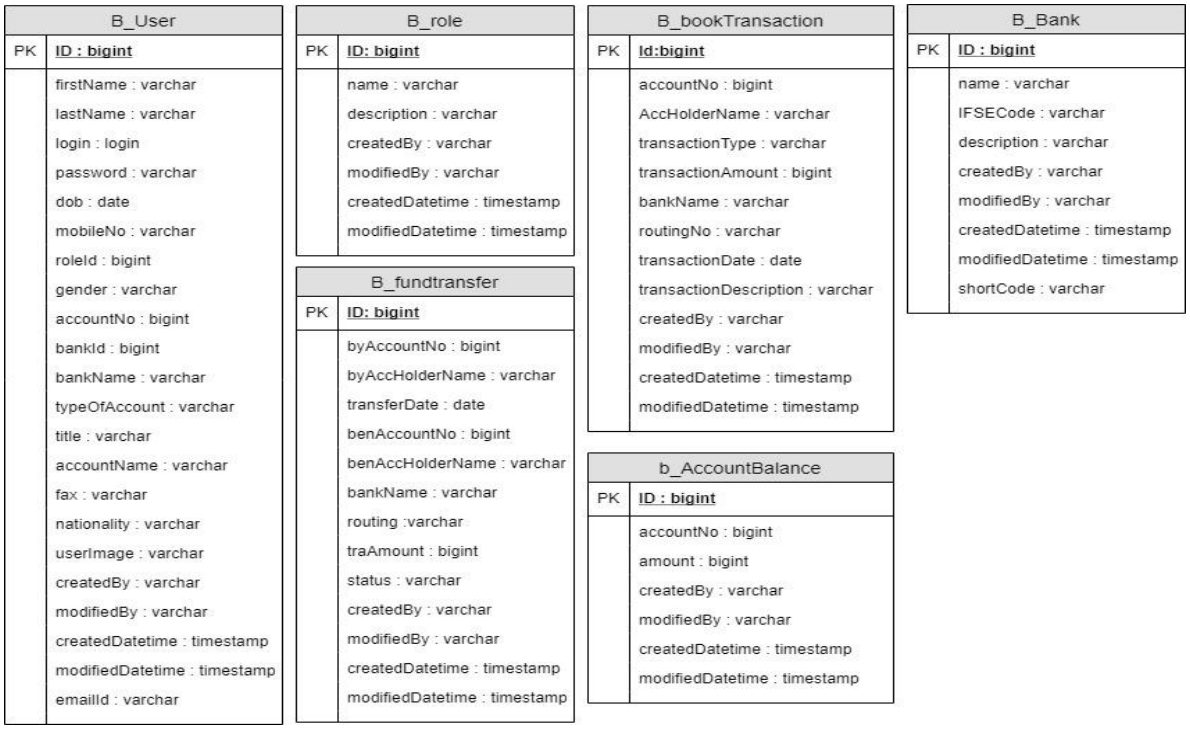


Рис 1.2.3

Діяльність щодо проектування / специфікації:

• Формулювання концепції.

• Розуміння проблем.

• Пропозиції високого рівня.

• Техніко-економічне обґрунтування.

• Розробка вимог.

• Архітектурне проектування.

ПРОЕКТУВАННЯ МОДУЛІВ

Діяльність щодо проектування / специфікації:

• Формулювання концепції.

• Розуміння проблеми.

• Техніко-економічне обґрунтування.

• Архітектурне проектування.

РОЗДІЛ 2

Дизайн модуля та проектування бази данних

2.1. Дизайн модуля Адмін

Адміністратор може ввійти в програму після підтвердження імені користувача та пароля.

ВХІД

Дизайн введення фокусується на контролері, необхідному обсязі введення, контролюючи помилки та недопустимі дані. Вхідні дані спроектовані таким чином, що забезпечують підтвердження та перевірку та. Дизайн вводу враховував такі речі:

о Перевірка вводу?

о Як слід упорядковувати чи кодувати дані?

о Повідомлення про перевірку має відображатися.

о Методи підготовки перевірок вхідних даних та кроки, які слід виконувати при виникненні помилки.

ВИХІД ДИЗАЙН

Вихід без помилок - це той, який відповідає вимогам користувача системи та чітко подає інформацію детально. У проекті вихідних даних визначається, яким чином інформація повинна бути переміщена для запиту. Це найважливіші та прямі вихідні дані для користувача.

Проектування комп’ютерних вихідних даних повинно проходити організовано або у форматі; правильний вихідний результат повинен відображатися, забезпечуючи при цьому, що кожен вихідний елемент розроблений так що люди знайдуть систему, можна використовувати простий та ефективний спосіб. При аналізі проектування комп'ютерних вихідних даних вони повинні:

Без помилок.

Без логічної помилки.

Формат виводу повинен бути правильним і зрозумілим для нових користувачів.

2.2 Розробка бази данних

База даних - це організований механізм, який має можливість зберігати інформацію, завдяки якій користувач може ефективно та ефективно отримувати збережену інформацію. Дані призначені для будь-якої бази даних і повинні бути захищені.

Дизайн бази даних - це дворівневий процес. На першому кроці вимоги користувачів збираються разом і створюється база даних, яка буде максимально чітко відповідати цим вимогам. Цей етап називається проектуванням інформаційного рівня, і він береться незалежно від будь-якої окремої СУБД.

На другому кроці цей проект інформаційного рівня переноситься у проект для конкретної СУБД, який буде використовуватися для впровадити відповідну систему. Цей етап називається проектуванням фізичного рівня, що стосується характеристик конкретних СУБД, які будуть

використовуватися. Дизайн бази даних працює паралельно з дизайном системи. Організація даних у базі даних спрямована на досягнення наступних двох основних цілей.

Цілісність даних
independence Незалежність даних

Нормалізація - це процес декомпозиції атрибути в додатку, результатом чого є набір таблиць з дуже простою структурою. Мета нормалізації - максимально спростити таблиці. Нормалізація здійснюється в цій системі з наступних причин.

• Щоб структурувати дані так, щоб не повторювати дані, це допомагає зберегти.

• Дозволити простий пошук даних у відповідь на запит та запит звіту.

• Спростити обслуговування даних за допомогою оновлення, вставки та видалення.

• Щоб зменшити необхідність реструктуризації або реорганізації даних, що виникають перед новими вимогами до програми.

СИСТЕМА УПРАВЛІННЯ БЕЗКОШТОВНИМИ БАЗАМ ДАНИХ (RDBMS): Реляційна модель представляє базу даних як сукупність відносин. Кожне відношення нагадує таблицю значень або файл записів. У офіційній термінології реляційної моделі рядок називається кортежем, заголовок стовпця - атрибутом, а таблиця - відношенням. Реляційна база даних складається з набору таблиць, кожному з яких присвоєно унікальне ім'я. Рядок у казці представляє набір пов’язаних значень.

2.3. Відносини домена і атрибута

Таблиця - це відношення. Рядки в таблиці називаються кортежами. Кортеж - це впорядкований набір з n елементів. Стовпці називаються атрибутами. Взаємозв'язки встановлені між кожною таблицею бази даних. Це забезпечує цілісність як референційних, так і цілісних відносин. Домен D - це набір атомних значень. Поширеним методом визначення домену є визначення типу даних, з якого витягуються значення даних, що утворюють домен. Також корисно вказати ім'я домену, щоб допомогти в інтерпретації його значень. Кожне значення у відношенні є атомним, яке не можна розкласти.

ВІДНОСИНИ:

Зв’язки таблиці встановлюються за допомогою ключа. Два основні ключі першочергового значення - це Первинний ключ та Зовнішній ключ. Цілісність сутності цілісності та референційна цілісність можуть бути встановлені за допомогою цих ключів. Цілісність цілісності передбачає, що жоден первинний ключ не може мати нульових значень.
Довідкова цілісність для кожного окремого значення зовнішнього ключа, в тому самому домені має існувати відповідне значення первинного ключа. Інший ключ - це Супер ключ та Ключі-кандидати.
Зв’язки встановлені між кожною таблицею бази даних. Це гарантує цілісність як довідкових, так і цілосних відносин.

НОРМАЛІЗАЦІЯ:

Як випливає з назви, це означало наведення речей у звичайній формі. Розробник додатків за допомогою нормалізації намагається досягти розумної організації даних у належні таблиці та стовпці та де користувач може легко співвіднести імена з даними. Нормалізація усуває повторювані групи даних і

тим самим уникає надмірності даних, що виявляється великим навантаженням на комп'ютерні ресурси. Сюди входять:

Нормалізація даних.

Виберіть власні імена для таблиць і стовпців.

Виберіть належну назву для даних.

Перша звичайна форма:

Перша нормальна форма стверджує, що домен атрибута повинен включати лише атомні значення і що значення будь-якого атрибута в кортежі має бути єдиним значенням із домену цього атрибута. Іншими словами, 1NF забороняє "відносини всередині відносин" або "відносини як значення атрибутів у кортежах". Єдиними значеннями атрибутів, дозволеними 1NF, є одиничні атомарні або неподільні значення.

Першим кроком є розміщення даних у Першій звичайній формі. Це може бути донором, перемістивши дані в окремі таблиці, де дані однакового типу в кожній таблиці. Кожній таблиці надається первинний ключ або зовнішній ключ відповідно до вимог проекту. У цьому ми формуємо нові відносини для кожного неатомного атрибута або вкладеного відношення. Це усунуло повторювані групи даних.

Зв’язок, як кажуть, знаходиться у першій нормальній формі, лише якщо воно відповідає обмеженням, що містять лише первинний ключ.

Друга звичайна форма:

Відповідно до Другої звичайної форми, для відносин, де первинний ключ містить кілька атрибутів, жоден атрибут nonkey не повинен функціонально залежати від частини первинного ключа.

У цьому розділі ми розкладаємо та встановлюємо нове відношення для кожного часткового ключа з його залежними атрибутами. Обов’язково

зберігайте зв’язок із вихідним первинним ключем та будь-якими атрибутами, які повністю функціонально залежать від нього. Цей крок допомагає витягувати дані, які залежать лише від окремого ключа.

Кажуть, що відношення має другу нормальну форму тоді і лише тоді, коли воно задовольняє всім умовам першої нормальної форми для атрибути первинного ключа та всіх непервинних ключів відношення повністю залежать лише від його первинного ключа.
Третя звичайна форма:

Відповідно до третьої звичайної форми, відношення не повинно мати атрибут nonkey, який функціонально визначається іншим атрибутом nonkey або набором атрибутів nonkey. Тобто, не повинно бути перехідної залежності від первинного ключа.

Ми розкладаємо та встановлюємо відношення, яке включає атрибути nonkey, що функціонально визначає інші атрибути nonkey. Цей крок зроблений для позбавлення від усього, що не цілком залежить від первинного ключа.

Відносини, як кажуть, знаходяться у третій нормальній формі, якщо лише у другій нормальній формі та більше над неключовими атрибутами відношення не повинен залежати від інших неключових атрибутів.

ВПРОВАДЖЕННЯ ТЕСТУВАННЯ СИСТЕМИ

Реалізація - це етап проекту, де теоретичний проект перетворюється на робочу систему. Це можна вважати найважливішим етапом досягнення успішної нової системи, щоб отримати впевненість користувачів у тому, що нова система буде працювати і буде ефективною та точною. Це в першу чергу стосується навчання та документації користувачів. Перетворення

зазвичай відбувається приблизно в той самий час, коли користувач проходить навчання або пізніше. Впровадження просто означає приведення в дію нового дизайну системи, що є процесом перетворення нового переглянутого дизайну системи в операційний.

РОЗДІЛ 3

Тестування системи. План тесту

Програмне забезпечення Тестування - це процес керованого запуску програмного забезпечення для того, щоб відповісти на питання - чи поводиться програмне забезпечення, як зазначено? Тестування програмного забезпечення часто використовується у поєднанні з термінами перевірки та перевірки.Перевірка - це перевірка або тестування елементів, що включає програмне забезпечення, на відповідність та узгодженість із відповідними специфікаціями. Тестування програмного забезпечення - це лише один із видів перевірки, який також використовує такі методи, як огляди, аналіз, перевірки та покрокові інструкції. Перевірка - це процес перевірки того, що вказане - це те, що насправді хотів користувач.

Тестування програмного забезпечення не слід плутати з налагодженням. Налагодження - це процес аналізу та локалізації помилок, коли програмне забезпечення не працює належним чином. Хоча ідентифікація деяких помилок буде очевидною з ігор із програмним забезпеченням, методичний підхід до тестування програмного забезпечення є набагато більш ретельним засобом для виявлення помилок. Тому налагодження - це діяльність, яка підтримує тестування, але не може замінити тестування.

Інші дії, які часто пов’язані з тестуванням програмного забезпечення, - це статичний аналіз та динамічний аналіз. Статичний аналіз досліджує вихідний код програмного забезпечення, шукає проблеми та збирає показники без фактичного виконання коду.

Динамічний аналіз

розглядає поведінку програмного забезпечення під час його виконання, щоб надати таку інформацію, як сліди виконання, хронологічні профілі та інформація про охоплення тесту.

Тестування - це набір заходів, який можна планувати в перспективі та проводити систематично. Тестування розпочинається на рівні модуля і спрямовується на інтеграцію всієї комп'ютерної системи. Без тестування ніщо не обходиться, оскільки для життєвого успіху цілей тестування системи існує кілька правил, які можуть служити цілями тестування. Вони

Тестування - це процес запуску програми з наміром знайти помилку. Хороший тестовий випадок - це той, який має велику можливість знайти невиявлену помилку. Успішний тест - це той, що виявляє невиявлену помилку.

Якщо тестування проводиться успішно відповідно до цілей, як зазначено вище, це дозволить виявити помилки в програмному забезпеченні, а також тестування демонструє, що функція програмного забезпечення, здається, працює відповідно до специфікації, що ця вимога до продуктивності виконана.

Існує три способи тестування програми.

- Для коректності
- Для ефективності реалізації
- Для обчислювальної складності

Тест на правильність повинен перевірити, чи програма виконує саме те, що вона була створена. Це набагато складніше, ніж це може здатися спочатку, особливо для великих програм.

ПЛАН ТЕСТУ

План тестування передбачає серію бажаного курсу дій, якого слід дотримуватися при здійсненні різних методів тестування. План випробувань діє як синій відбиток для дії, якій слід слідувати. Інженери-програмісти створюють комп'ютерну програму, її документацію та відповідні структури даних.Розробники програмного забезпечення завжди відповідають за тестування окремих підрозділів програм, гарантуючи, що кожен виконує функцію, для якої він був розроблений. Існує незалежна тестова група (ITG), яка має усунути невід’ємні проблеми, пов’язані з тим, щоб дозволити будівельнику перевірити побудовану річ. Конкретні цілі тестування повинні бути зазначені у вимірюваних термінах. Таким чином, середній час до відмови, витрати на пошук та виправлення дефектів, залишкову щільність дефектів або частоту їх виникнення та робочі години тесту на регресійний тест повинні бути зазначені в плані тестування.

Рівні тестування включають:

- модульне тестування
- тестування інтеграції
- тестування перевірки даних
- тестування вихідних даних

ТЕСТУВАННЯ ОДИНИЦІ

Блокове тестування зосереджує зусилля перевірки на найменшій одиниці програмного забезпечення - програмному компоненті або модулі. Використовуючи опис дизайну рівня компонента як керівництво, важливі шляхи керування перевіряються для виявлення помилок у межах модуля. Відносна складність випробувань та нерозкритий обсяг, встановлений для модульних випробувань. Модульне тестування орієнтоване на білу скриньку, і крок може виконуватися паралельно для кількох компонентів.

Модульний інтерфейс тестується, щоб переконатись, що інформація належним чином надходить у тестовий блок і з нього. Локальна структура даних досліджується, щоб переконатися, що дані, що зберігаються, тимчасово зберігають свою цілісність на всіх етапах виконання алгоритму. Граничні умови перевіряються, щоб гарантувати, що всі оператори в модулі були виконані принаймні один раз. Нарешті, перевіряються всі шляхи обробки помилок.

Перевірка потоку даних через інтерфейс модуля необхідна перед початком будь-якого іншого тестування. Якщо дані не надходять і не виходять належним чином, усі інші тести несумісні. Вибіркове тестування шляхів виконання є важливим завданням під час модульного тестування. Хороший дизайн передбачає передбачення умов помилок та налаштування шляхів обробки помилок для перенаправлення або чистого припинення обробки, коли помилка все-таки виникає. Тестування меж є останнім завданням етапу модульного тестування. Програмне забезпечення часто виходить з ладу на своїх межах.

Модульне тестування проводилось у системі Sell-Soft, розглядаючи кожен модуль як окрему сутність та тестуючи кожен із них із широким спектром

тестових входів. Було виявлено та виправлено деякі недоліки внутрішньої логіки модулів.

3.2. Тестування інтеграції

Інтеграційне тестування - це систематизований прийом для побудови структури програми, одночасно проводячи тести для виявлення помилок, пов'язаних із взаємодією. Мета полягає в тому, щоб взяти модульно перевірені компоненти та побудувати структуру програми, яка була продиктована дизайном. Вся програма перевірена як ціла. Виправлення складно, оскільки ізоляція причин ускладнюється величезним простором усієї програми. Як тільки ці помилки виправляються, з'являються нові, і процес продовжується, здавалося б, нескінченним циклом.

Після модульного тестування в системі Sell-Soft всі модулі були інтегровані для перевірки на наявність невідповідностей в інтерфейсах . Окрім того, були усунені відмінності в структурах програм, і було створено унікальну структуру програми.

3.3. Тестування валідації та системи

Це останній крок у тестуванні. У цьому вся система була протестована в цілому з усіма формами, кодом, модулями та модулями класів. Ця форма тестування в народі відома як тестування Black Box або тестування системи.

Метод тестування Black Box фокусується на функціональних вимогах програмного забезпечення. Тобто тестування Black Box дозволяє інженеру програмного забезпечення вивести набори умов введення, які повністю виконуватимуть усі функціональні вимоги програми.

Тестування Black Box намагається знайти помилки в наступних категоріях; неправильні або відсутні функції, помилки інтерфейсу, помилки в структурах

даних або зовнішньому доступі до даних, помилки продуктивності та помилки ініціалізації та помилки припинення.

ТЕСТУВАННЯ ВИХІДУ ТА ТЕСТУВАННЯ ПРИЙНЯТТЯ КОРИСТУВАЧЕЮ

Розглянута система перевірена на прийняття користувачами; тут він повинен задовольнити потреби фірми. Програмне забезпечення має підтримувати зв'язок із перспективною системою; користувач на момент
розробки та внесення змін, коли це потрібно. Це зроблено щодо таких пунктів:

Дизайн вхідних екранів,

Проектування вихідних екранів,

Інтернет-повідомлення для керівництва користувачем тощо.

Наведене вище тестування проводиться з використанням різних видів даних тесту. Підготовка тестових даних відіграє життєво важливу роль у тестуванні системи. Після підготовки тестових даних досліджувана система тестується з використанням цих тестових даних. Під час тестування системи, за допомогою якої помилки тестових даних знову виявляються та виправляються за допомогою вищезазначених кроків тестування, а також виправлення також зазначаються для подальшого використання.

НАВЧАННЯ

Один раз система успішно розроблена наступним важливим кроком є забезпечення належної підготовки адміністраторів до роботи з системою.

Це тому, що успіх системи незмінно залежить від того, як вони експлуатуються та використовуються. Реалізація залежить від того, як потрібні люди опиняться в потрібному місці в потрібний час. Освіта передбачає створення відповідної атмосфери та мотивацію користувача. Адміністратори ознайомлені з процедурами запуску системи, що постійно проробляють послідовність дій.

Реалізація - це стан у проекті, коли теоретичний дизайн перетворюється на робочу систему . Цим користувачі отримують впевненість, що система буде працювати ефективно. Система може бути впроваджена лише після тестування.

Персонал системи перевіряє можливість використання системи. Фактичні дані були введені в систему, і робота системи була ретельно відстежена. Параметр master був обраний з головного меню, а фактичні дані вводились через відповідні екрани введення. Потім було вивчено переміщення даних, після чого було обрано правильний варіант запитів, який містить різні звіти. Службові програми надають різні дані, необхідні для введення інвентаризації та тестування модуля. Отримані задовільні результати. Звіти, пов’язані з цими процесами, також були успішно сформовані. Різні формати екрану введення перераховані в додатку.

Покрокові інструкції з впровадження гарантують, що завершена система насправді вирішує початкову проблему. Це покрокове керівництво відбувається безпосередньо перед введенням системи в експлуатацію, і воно повинно включати ретельний перегляд усіх посібників, навчальних матеріалів та системної документації. Знову ж таки, користувачі, аналітик та співробітники комп'ютерних служб можуть брати участь у цій зустрічі.

3.4. Для чого STS

Не доводиться сумніватися в тому, що Spring є найпопулярнішим Java фреймворком, а Spring Tools спрощує створення проектів на Spring і Spring Boot в Eclipse. Використовуючи STS, ви можете швидко створювати проекти на Spring Boot, використовуючи просту інтеграцію start.spring.io. Він також підтримує розробку додатків з використанням Spring Java-Config, розширене автодоповнення коду, content-assist, валідацію і підтримку quick-fix для додатків на Spring. Він ідеально підходить для розробки мікросервісов з використанням Spring, оскільки дозволяє інтегрувати IDE для Cloud Foundry, включаючи налагодження в хмарі.

STS - це просто заздалегідь сконфигурированное затемнення. це абсолютно безкоштовно для будь-яких цілей. Плюси і мінуси використання STS йдуть рука об руку: в ньому встановлені деякі плагіни, які можуть бути корисні або просто уповільнюють вашу IDE. У ньому є речі, які можуть вам знадобитися і які не потрібно завантажувати, але у мене можуть бути речі, які вам не знадобляться. Наскільки мені відомо, ви можете встановити всі плагіни, які є у STS в вашому eclipse.

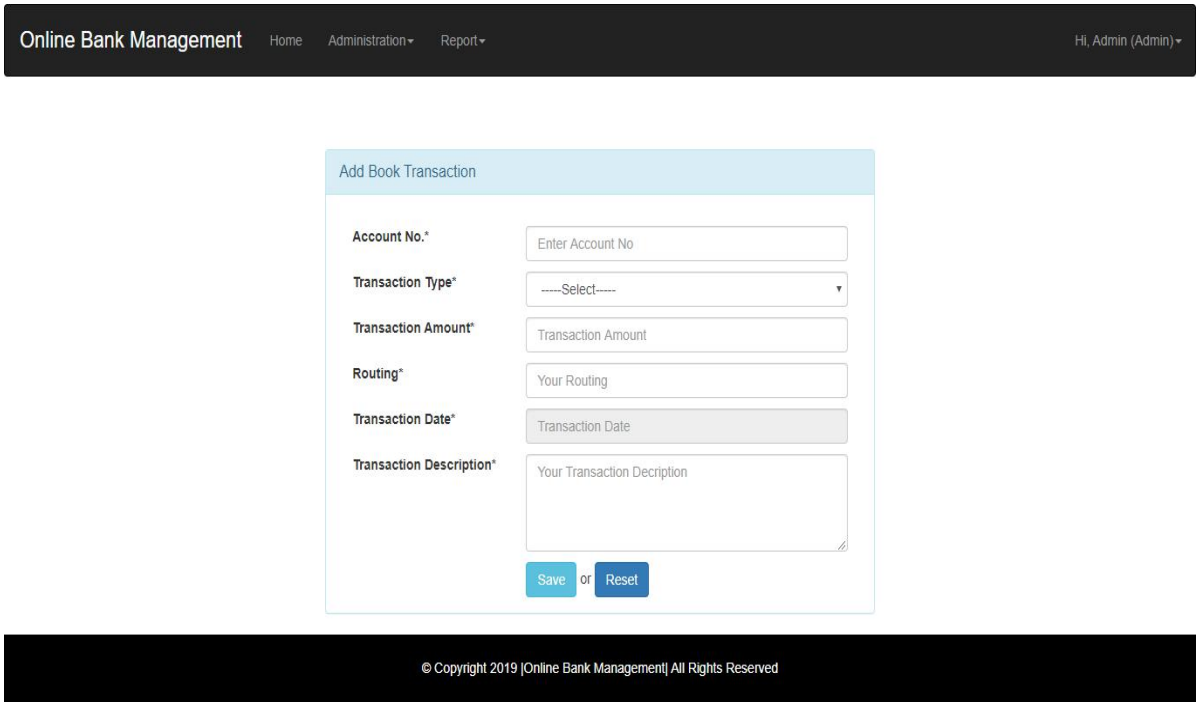
Інтеграція Maven для Eclipse

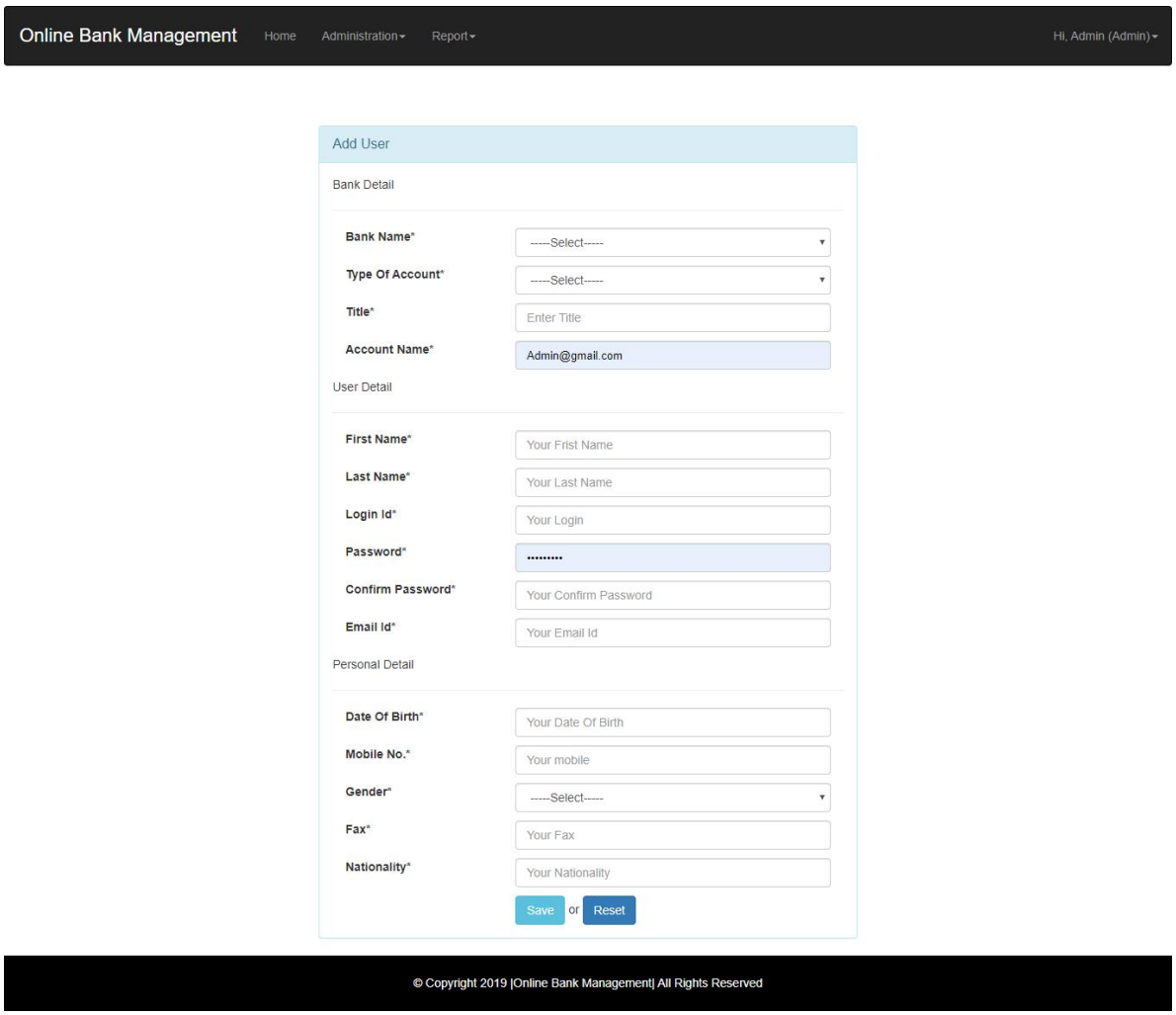
Плагін M2E або плагін Maven Integration for Eclipse - це ще один популярний плагін Eclipse, необхідний для розробки на Java. Він забезпечує комплексну інтеграцію Maven для Eclipse. Ви можете використовувати M2E для управління як простими, так і мульти-модульними проектами Maven, виконувати збірки Maven через інтерфейс Eclipse і взаємодіяти з репозиторіями Maven. Крім того, деякі плагіни залежать від того, яку версію Eclipse ви використовуєте: існує окремий плагін для Juno, Luna і так далі.

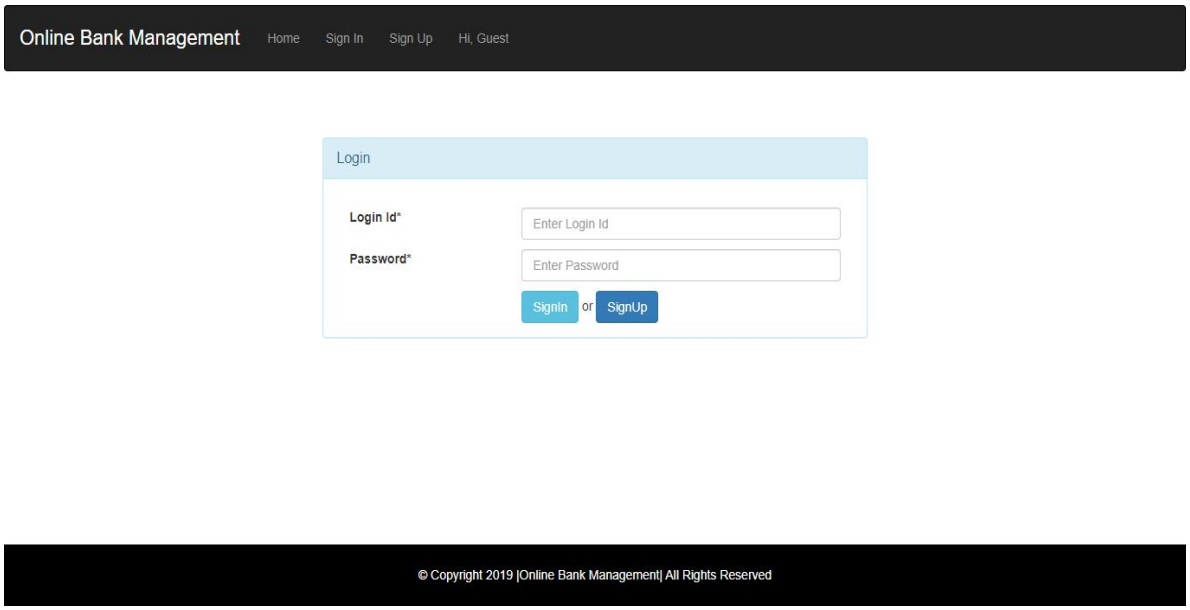
Обзор плафторми Eclipse

Eclipse під контролем організації Eclipse Foundation. Eclipse це програма з відкритим кодом яка написана на мові програмування Java,основна мета це підвищення продуктивності процесу. Претендує на статус найбільш популярною Java IDE і єдиним конкурентом такої потужної платформи як NetBeans. Але на відміну від NetBeans який для створення елемента призначеного для користувача інтерфейсу використовує від платформи незалежну бібліотеку Swing, Eclipse використовує платформо-залежна бібліотека SWT Standard Widget Toolkit.

3.5. Скріншоти програми




Online Bank Management
Home
Administration
Report
Hi, Admin (Admin)
Add User
Bank Detail
Bank Name*
-----Select-----
Type Of Account*
-----Select-----
Title*
Enter Title
Account Name*
Admin@gmail.com
User Detail
First Name*
Your Frist Name
Last Name*
Your Last Name
Login Id*
Your Login
Password*
Confirm Password*
Your Confirm Password
Email Id*
Your Email Id
Personal Detail
Date Of Birth*
Your Date Of Birth
Mobile No.*
Your mobile
Gender*
-----Select-----
Fax*
Your Fax
Nationality*
Your Nationality
Save
or
Reset
© Copyright 2019 |Online Bank Management| All Rights Reserved


Online Bank Management
Home
Sign In
Sign Up
Hi, Guest
Login
Login Id*
Enter Login Id
Password*
Enter Password
Signin
or
SignUp
© Copyright 2019 |Online Bank Management| All Rights Reserved

3.6. Код

Controller

```
package in.co.online.bank.mgt.controller;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.log4j.Logger;

import in.co.online.bank.mgt.bean.AccountBalanceBean;
import in.co.online.bank.mgt.bean.BankBean;
import in.co.online.bank.mgt.bean.UserBean;
import in.co.online.bank.mgt.exception.ApplicationException;
import in.co.online.bank.mgt.model.AccountBalanceModel;
import in.co.online.bank.mgt.model.BankModel;
import in.co.online.bank.mgt.model.BookTransactionModel;
import in.co.online.bank.mgt.util.DataUtility;
import in.co.online.bank.mgt.util.ServletUtility;

/**
 * Servlet implementation class AccountBalanceCtl
 */
@WebServlet(name = "AccountBalanceCtl", urlPatterns =
```

```
{ "/ctl/AccountBalanceCtl" })
public class AccountBalanceCtl extends BaseCtl {
	private static final long serialVersionUID = 1L;

	private static Logger log=Logger.getLogger(AccountBalanceCtl.class);

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		log.debug("AccountBalanceCtl doGet method start");

		AccountBalanceModel model = new AccountBalanceModel();

			HttpSession session=request.getSession();
			UserBean uBean=(UserBean) session.getAttribute("user");

				AccountBalanceBean bean;
				try {
					bean =
model.findByAcoountNo(uBean.getAccountNo());
					ServletUtility.setOpration("Edit", request);
					ServletUtility.setBean(bean, request);
				} catch (ApplicationException e) {
					ServletUtility.handleException(e, request,
response);
					return;
```

```
                    }


                        ServletUtility.forward(getView(), request, response);
                        log.debug("AccountBalanceCtl doGet method end");
        }

        @Override
        protected String getView() {
                // TODO Auto-generated method stub
                return OBMView.ACCOUNT_BALANCE_VIEW;
        }

}
```

Database Madel

```
package in.co.online.bank.mgt.model;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import org.apache.log4j.Logger;

import in.co.online.bank.mgt.bean.AccountBalanceBean;
import in.co.online.bank.mgt.bean.RoleBean;
```

```
import in.co.online.bank.mgt.exception.ApplicationException;
import in.co.online.bank.mgt.exception.DatabaseException;
import in.co.online.bank.mgt.exception.DuplicateRecordException;
import in.co.online.bank.mgt.util.JDBCDataSource;

public class AccountBalanceModel {

        private static Logger log = Logger.getLogger(AccountBalanceModel.class);

        /**
         * Find next PK of Role
         *
         * @throws DatabaseException
         */
        public Integer nextPK() throws DatabaseException {
                log.debug("Model nextPK Started");
                Connection conn = null;
                int pk = 0;
                try {
                        conn = JDBCDataSource.getConnection();
                        PreparedStatement pstmt = conn.prepareStatement("SELECT
MAX(ID) FROM B_AccountBalance");
                        ResultSet rs = pstmt.executeQuery();
                        while (rs.next()) {
                                pk = rs.getInt(1);
                        }
                        rs.close();
                } catch (Exception e) {
                        log.error("Database Exception..", e);
```

```
                throw new DatabaseException("Exception : Exception in
getting PK");
            } finally {
                JDBCDataSource.closeConnection(conn);
            }
            log.debug("Model nextPK End");
            return pk + 1;
        }

        public long add(AccountBalanceBean bean) throws ApplicationException,
DuplicateRecordException {
            log.debug("Model add Started");
            Connection conn = null;
            int pk = 0;

            try {
                conn = JDBCDataSource.getConnection();
                pk = nextPK();

                // Get auto-generated next primary key
                System.out.println(pk + " in ModelJDBC");
                conn.setAutoCommit(false); // Begin transaction
                PreparedStatement pstmt = conn.prepareStatement("INSERT
INTO B_AccountBalance VALUES(?,?,?,?,?,?,?)");
                pstmt.setInt(1, pk);
                pstmt.setLong(2, bean.getAccountNo());
                pstmt.setLong(3, bean.getAmount());
                pstmt.setString(4, bean.getCreatedBy());
                pstmt.setString(5, bean.getModifiedBy());
```

```
                pstmt.setTimestamp(6, bean.getCreatedDatetime());
                pstmt.setTimestamp(7, bean.getModifiedDatetime());
                pstmt.executeUpdate();
                conn.commit(); // End transaction
                pstmt.close();
            } catch (Exception e) {
                e.printStackTrace();
                log.error("Database Exception..", e);
                try {
                    conn.rollback();
                } catch (Exception ex) {
                    throw new ApplicationException("Exception : add
rollback exception " + ex.getMessage());
                }
                throw new ApplicationException("Exception : Exception in add
AccountBalance");
            } finally {
                JDBCDataSource.closeConnection(conn);
            }
            log.debug("Model add End");
            return pk;
        }

        public AccountBalanceBean findByPK(long pk) throws
ApplicationException {
            log.debug("Model findByPK Started");
            StringBuffer sql = new StringBuffer("SELECT * FROM
B_AccountBalance WHERE ID=?");
            AccountBalanceBean bean = null;
```

```
        Connection conn = null;
        try {
            conn = JDBCDataSource.getConnection();
            PreparedStatement pstmt =
conn.prepareStatement(sql.toString());
            pstmt.setLong(1, pk);
            ResultSet rs = pstmt.executeQuery();
            while (rs.next()) {
                bean = new AccountBalanceBean();
                bean.setId(rs.getLong(1));
                bean.setAccountNo(rs.getLong(2));
                bean.setAmount(rs.getLong(3));
                bean.setCreatedBy(rs.getString(4));
                bean.setModifiedBy(rs.getString(5));
                bean.setCreatedDatetime(rs.getTimestamp(6));
                bean.setModifiedDatetime(rs.getTimestamp(7));
            }
            rs.close();
        } catch (Exception e) {
            log.error("Database Exception..", e);
            throw new ApplicationException("Exception : Exception in
getting AccountBalance by pk");
        } finally {
            JDBCDataSource.closeConnection(conn);
        }
        log.debug("Model findByPK End");
        return bean;
    }
```

```
	public AccountBalanceBean findByAcoountNo(long pk) throws
ApplicationException {
		log.debug("Model findByPK Started");
		StringBuffer sql = new StringBuffer("SELECT * FROM
B_AccountBalance WHERE AccountNo=?");
		AccountBalanceBean bean = null;
		Connection conn = null;
		try {
			conn = JDBCDataSource.getConnection();
			PreparedStatement pstmt =
conn.prepareStatement(sql.toString());
			pstmt.setLong(1, pk);
			ResultSet rs = pstmt.executeQuery();
			while (rs.next()) {
				bean = new AccountBalanceBean();
				bean.setId(rs.getLong(1));
				bean.setAccountNo(rs.getLong(2));
				bean.setAmount(rs.getLong(3));
				bean.setCreatedBy(rs.getString(4));
				bean.setModifiedBy(rs.getString(5));
				bean.setCreatedDatetime(rs.getTimestamp(6));
				bean.setModifiedDatetime(rs.getTimestamp(7));
			}
			rs.close();
		} catch (Exception e) {
			log.error("Database Exception..", e);
			throw new ApplicationException("Exception : Exception in
getting AccountBalance by pk");
		} finally {
```

```
                JDBCDataSource.closeConnection(conn);
            }
            log.debug("Model findByPK End");
            return bean;
        }

        public void update(AccountBalanceBean bean) throws
ApplicationException, DuplicateRecordException {
            log.debug("Model update Started");
            Connection conn = null;

            try {
                conn = JDBCDataSource.getConnection();
                conn.setAutoCommit(false); // Begin transaction
                PreparedStatement pstmt = conn.prepareStatement(
                        "UPDATE B_AccountBalance SET
AccountNo=?,Amount=?,CREATEDBY=?,MODIFIEDBY=?,CREATEDDATET
IME=?,MODIFIEDDATETIME=? WHERE ID=?");
                pstmt.setLong(1, bean.getAccountNo());
                pstmt.setLong(2, bean.getAmount());
                pstmt.setString(3, bean.getCreatedBy());
                pstmt.setString(4, bean.getModifiedBy());
                pstmt.setTimestamp(5, bean.getCreatedDatetime());
                pstmt.setTimestamp(6, bean.getModifiedDatetime());
                pstmt.setLong(7, bean.getId());
                pstmt.executeUpdate();
                conn.commit(); // End transaction
                pstmt.close();
            } catch (Exception e) {
```

```
                    log.error("Database Exception..", e);
                    try {
                        conn.rollback();
                    } catch (Exception ex) {
                        throw new ApplicationException("Exception : Delete
rollback exception " + ex.getMessage());
                    }
                    throw new ApplicationException("Exception in updating Role
");
                } finally {
                    JDBCDataSource.closeConnection(conn);
                }
                log.debug("Model update End");
        }

}
```

Hibernate configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/Bank_mgt_hib</pr
operty>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">root</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>


        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>


        <property name="hibernate.c3p0.min_size">10</property>
        <property name="hibernate.c3p0.max_size">100</property>
        <property name="hibernate.c3p0.acquire_increment">10</property>
```

```
        <property name="hibernate.c3p0.timeout">10</property>
        <property name="hibernate.c3p0.">10</property>


        <mapping class="in.co.online.bank.mgt.bean.UserBean" />
        <mapping class="in.co.online.bank.mgt.bean.RoleBean" />
        <mapping class="in.co.online.bank.mgt.bean.AccountBalanceBean"
/>
        <mapping class="in.co.online.bank.mgt.bean.BankBean" />
        <mapping
            class="in.co.online.bank.mgt.bean.BookTransactionBean" />
        <mapping class="in.co.online.bank.mgt.bean.FundTransferBean" />



    </session-factory>
</hibernate-configuration>
```

Висновок

"Система управління банківською діяльністю" веде щоденні підрахунки як повна банківська справа. Вона може зберігати інформацію про тип рахунку, форму відкриття рахунку, депозит, зняття коштів, і Пошук транзакції, звіт про транзакції, форма відкриття індивідуального рахунку, груповий рахунок. Цікавою частиною цього проекту є: він відображає звіти про транзакції, статистичний звіт про тип рахунку та інформацію про відсотки.

Список використаних джерел

БІБЛІОГРАФІЯ

КНИГИ:

1. Чарльз Хемпфед (2000), Університет Visual Basic, Торонто
2. Герберт Шильдт (2000) "Visual Basic 6.0" Тата Макгроу Хілл
3. Джон Зуковскі (2000) "Visual Basic 6,0" "Публікації ВРВ
4. Джеймі Яворський" Visual Basic 6,0 "Техмедіа
5. Стефен Деннінгер, «Visual Basic 6.0», Авторська преса

6. Ян Сомервіль, «Програмна інженерія»

ОНЛАЙН-ДОВІДКА:

www.w3schools.com

www.theserverside.com

www.visual.com

1.2 Діаграми

https://stackoverflow.com/questions/21775740/stsspring-tool-suite-and-eclipse

https://www.lucidchart.com/pages/ru/erd-%D0%B4%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0

https://habr.com/ru/company/trinion/blog/340064/

База данних

https://metanit.com/sql/mysql/4.8.php

https://proselyte.net/tutorials/jdbc/simple-application-example/

https://www.w3schools.com/sql/sql_syntax.asp

3.4 STS

https://habr.com/ru/post/487796/