МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерних наук

ТЕРЛЕЦЬКИЙ Андрій Юрійович

Програмна система реалізації товарів на основі VDOM/ Software system of goods based on VDOM

напрям підготовки: 6.050103 - Програмна інженерія
фахове спрямування - Програмне забезпечення систем

Бакалаврська дипломна робота

Виконав студент групи ПЗС-41
А. Ю. Терлецький

_____

Науковий керівник:
викладач ВЕРЕМЧУК А.В.

_____

Бакалаврську дипломну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____ **А. В. Пукас**

ТЕРНОПІЛЬ - 2016

ЗМІСТ

ВСТУП

Компанія VDOM Box International займається розробкою і просуванням власного web-сервера додатків під назвою VDOM Box. Сервер поставляється в вигляді готового рішення, тобто об’єднує в собі апаратну і програмну частини.

Компанія також надає інструменти розробника для створення додатків під її платформу, відому під назвою VDOM.

Останнім часом популярність платформи VDOM зростає, відповідно зростає і число зацікавлених клієнтів. На даний момент компанія VDOM Box International активно укладає контракти з розповсюдження своєї продукції з дистриб’юторами в різних країнах. При такому масштабному розповсюдженні товарів, компанії, як і її партнерам, необхідно вести загальну базу клієнтів, реєструвати замовлення, відстежувати проходження товарів.

До недавнього часу, наведені вище завдання вирішувалися за допомогою системи підтримки продажів, реалізованої на платформі OpenERP. Однак неможливість інтеграції існуючої системи з іншими додатками, змусила компанію шукати інші рішення.

Метою даної роботи є розробка системи підтримки продажу товарів, яка з одного боку буде використовуватися підрозділами компанії VDOM Box International за допомогою web-інтерфейсу, а з іншого боку - інтегруватися з іншими додатками, розробленими компанією.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПІДТРИМКИ ПРОЦЕСІВ РЕАЛІЗАЦІЇ ТОВАРІВ ЧЕРЕЗ ІНТЕРНЕТ

1.1. Опис предметної області

Електронна комерція – придбання чи продаж товарів за допомогою електронних носіїв, або через мережу, подібну Internet.

Система електронної комерції – це комплекс програмно-апаратних і мережних засобів, що дозволяють організувати взаємодію між суб’єктами бізнес-процесів з допомогою електронних засобів обміну інформацією (у тому числі з використовуванням Інтернет технологій). Об’єкт електронної комерції – торгівля за допомогою засобів електронного обміну даними. Суб’єкти електронної торгівлі – фізичні та юридичні особи.

Поняття електронної комерції включає в себе замовлення, оплату та доставку товарів. Термін "електронна комерція" виник практично відразу вслід за появою ЕОМ в 50-й, 60-й роки. Одними з перших програмних додатків були програми для транспорту – замовлення квитків, обмін даними між різними службами під час підготовки рейсів.

Концепція обміну електронними повідомленнями:

1. Кожна організація має свої власні прикладні бізнес-системи (MRP (Material Requirements Planning) систему планування матеріальних витрат, або MRP II (Manufacturer Resource Planning) систему планування витрат виробництва, або ERP (Enterprise Resource Planning) систему управління підприємством).

2. Кожна організація має один EDI-шлюз, через який вона обмінюється стандартними повідомленнями зі всіма іншими організаціями.

У 80-х склалася ситуація співіснування 2-х систем стандартів – європейського і американського. І лише в с середині 90-х починається рух по об’єднанню двох стандартів. Причина - поява web-технологій. Невід’ємною

частиною web-технологій є використання web-браузерів – програмних продуктів, побудованих на використані гіпертекстового посилального механізму, в основі якого лежить мова гіпертекстової розмітки HTML – HyperText Markup Language. HTML дозволяє форматувати і оформляти Web-сторінки, створюючи у вікні Web-браузера інформаційне середовище з простим, інтуїтивно зрозумілим людині інтерфейсом.

Розвиток данної технологій і інтеграція Web-браузера з більшістю форматів представлення даних і прикладних програмами дає можливість зробити його єдиним програмним інструментом ведення бізнесу. Створюється стандарт EDIINT (EDIFACT over Internet). Сучасні системи електронного обміну даними використовують даний стандарт.

Інший стандарт обміну інформацією – XML (eXtensible Markup Language). XML не тільки містить інструкції для Web-браузера – як повинна виглядати сторінка, але і дозволяє визначати типи документів, що передаються.

Електронна торгівля включає в себе п’ять відносно-незалежних процесів:

– доступ до інформації;

– оформлення замовлення;

– виконання оплати;

– виконання замовлення;

– обслуговування і підтримка.

Доступ до інформації. Виконується прозоро та неупереджено, споживачі вільно отримують доступ до інформації про компанію та про її товари і послуги.

Надання такої інформації не має пріоритетів відносно аналогічної інформації конкурентів. Тому потрібна реклама та відповідна маркетингова політика.

Крім того прозорість доступу до інформації є підґрунтям до проведення маркетингових досліджень ринку. З’ясування ставлення до продукції тощо.

Оформлення замовлення. Виконується в електронному вигляді. Виконується аналогічно паперовому оформленню. Електронні форми для

заповнення інформації про замовлення можуть повністю дублювати вигляд паперових.

Рис. 1.1. Схема видів електронної комерції

Виконання оплати. Допустимі будь які форми та засоби сплати, що відповідають звичайній комерції. Крім того використовуються специфічні для електронної комерції способи – електронні системи сплати, зокрема цифрові гроші. Електронні системи сплати еквівалентні, наприклад, кредитним і дебетовим картам і чекам, але находяться на стадії свого розвитку. На сьогоднішній день єдиного стандарту не існує.

Виконання замовлення. Обмін інформацією виконується в єдиному інформаційному просторі. Обмін продукцією також може виконуватись в межах даного простору за однієї умови: продукт може бути представлений в електронному вигляді (програмне забезпечення, бази даних, будь що, що може мати електронний носій). В іншому випадку необхідна організація підрозділу, що буде займатись фактичною доставкою продукції, яка реалізується.

Обслуговування і підтримка. Будь яка інформація про продукцію, що реалізується, повинна бути вільно доступною для будь яких споживачів:

– технічна документація;

– рекомендації по використанню тощо.

Організація вільного обміну інформацією між клієнтами (як фактичними так і потенційними) – електронна пошта, форуми, дошки об’яв тощо.

Можливості електронної комерції зумовлюють:

– зростання конкуренції;

– глобалізація сфер діяльності;

– персоналізація взаємодії;

– скорочення каналів розповсюдження товарів;

– економія витрат.

Зростання конкуренції. Вільний доступ до інформації конкурентів підвищує рівень вимог до продукції з боку потенційних замовників. У відповідь на це підприємства змушені змінювати способи організації і управління бізнесом. Відбувається відмова від старої ієрархічної структури, зникають бар'єри між відділеннями компанії, спрощується взаємодія між компаніями. Підвищується конкурентоспроможність. Підприємства стають "ближчими до замовника". Замовник одержує більш високу якість обслуговування.

Глобалізація сфер діяльності. Електронний обмін інформацією значно змінює просторовий і часовий масштаби ведення комерції. Знімаються територіальні обмеження. Найдрібніші постачальники можуть досягати глобальної присутності і займатися бізнесом у світовому масштабі. Відстань між продавцем і покупцем важлива лише з точки зору транспортних витрат лише на етапі доставки товарів та їх сплати. Даний факт дозволяє ухвалювати бізнес рішення у декілька разів швидше у порівнянні з традиційним способом.

Персоналізація взаємодії. Використання засобів електронної взаємодії забезпечує одержання докладної інформації про запити кожного індивідуального замовника і автоматичне надання продуктів і послуг у відповідності до індивідуальних вимог.

Скорочення каналів розповсюдження товарів. Шлях товару від постачальника до замовника істотно скорочуються. Товари успішно доставляються безпосередньо від виробника споживачу, в обхід традиційних каналів у вигляді оптових і роздрібних складів. Вартість товарів стає більш дешевою.

Особливий випадок – продукти і послуги, які можуть бути доставлені електронним способом. При цьому шлях доставки скорочується максимально. Електронний спосіб широко застосовується для доставки цифрових продуктів індустрії розваг (фільми, відео, музика, журнали і газети), інформації, засобів навчання і особливо ефективно використовується компаніями, що займаються розробкою і поставкою програмного забезпечення.

Економія витрат. Будь–який процес, в якому можна використовувати електронну взаємодію, має потенціал для скорочення витрат, що, у свою чергу, може бути основою зниження цін для замовників (за рахунок усунення проміжних ланок).

3. Категорії електронної комерції. В залежності від сфери використання електронна комерція поділяється на наступні категорії:

1. бізнес – бізнес (B2B);
2. споживач – споживач (C2C);
3. бизнес – споживач (B2C);
4. бізнес – адміністрація (B2A);
5. споживач – адміністрація (C2A).

Бізнес – бізнес. Дана категорія включає в себе всі рівні взаємодії між компаніями. При цьому можуть використовуватися спеціальні технології і стандарти електронного обміну даними.

Споживач–споживач. Категорія допускає можливість взаємодії споживачів для обміну комерційною інформацією. Це може бути обмін досвідом придбання того або іншого товару, обмін досвідом взаємодії з тією або іншою фірмою тощо. До цієї ж області відноситься і форма аукціонної торгівлі між фізичними особами.

Бізнес–споживач. Дана категорія обумовлює характер електронної роздрібної торгівлі. Наприклад електронні магазини. Інтернет магазин – це web–сервер, що надає до продажу товари або послуги та забезпечує механізми їх замовлення та сплати.

Бізнес–адміністрація. Взаємодія бізнесу і адміністрації включає ділові зв'язки комерційних структур з державними організаціями, починаючи від місцевої влади і закінчуючи міжнародними організаціями.

Споживач–адміністрація. Дана категорія якнайменше розвинута, проте має достатньо високий потенціал, який може бути використаний для організації взаємодії державних структур і споживачів, особливо в соціальній і податковій сфері.

Приклади реалізації можливостей електронного комерції. Віртуальні аукціони. Віртуальні аукціони дають користувачам можливість не тільки реалізовувати товари і послуги через Інтернет, але і проводити їхні тестові продажі. Компанії можуть використовувати віртуальні аукціони як інструмент маркетингової оцінки, за допомогою якого можна визначити величину первинного попиту і ринкову ціну для нового продукту.

Інтернет–трейдінг. Суб'єкти електронної торгівлі які дозволяють клієнтам (банками або брокерськими компаніями) здійснювати покупку/продаж цінних паперів та валюти в реальному часі використовуючи мережу електронного обміну даними.

Інтернет–банкінг. Сфера електронної комерції, що створюється та підтримується банківськими структурами і призначена для здійснення через мережу всіх стандартних банківських операції, які можуть бути виконані клієнтом в офісі банку, за винятком операцій з готівкою.

Перспективи використання Інтернет–банкінгу:

- здійснення комунальних платежів;
- виконання грошових переказів на будь–який рахунок в будь–якому іншому банку;
- виконання грошових переказів в оплату рахунків за товари, у тому числі куплені через web–магазини;
- здійснення операцій купівлі та продажу іноземної валюти;
- ведення рахунку пластикової карти;
- відкриття рахунку, одержання інформації про його стан тощо;

одержання інших послуги: підписку на журнали і газети тощо.

1.2. Огляд і аналіз існуючих аналогів, що реалізують функції предметної області

Система керування вмістом (СКВ; англ. Content Management System, CMS) — програмне забезпечення для організації веб-сайтів чи інших інформаційних ресурсів в Інтернеті чи окремих комп'ютерних мережах.

Існують сотні, а може, навіть й тисячі доступних CMS — систем. Завдяки їхній функціональності їх можна використовувати в різних компаніях. Незважаючи на широкий вибір інструментальних та технічних засобів, наявних в CMS, існують загальні для більшості типів систем характеристики.

Перші СКВ були розроблені у великих корпораціях для організації роботи з документацією. У 1995-му від компанії CNET відокремилася окрема компанія Vignette, яка започаткувала ринок для комерційних СКВ. З часом діапазон продукції розширювався і все більше інтегрувався у сучасні мережеві рішення аж до популярних веб-порталів.

Багато сучасних СКВ поширюються як безкоштовні і легкі у встановленні (інсталяції) програми, які розробляються групами ентузіастів під ліцензієюGNU/GPL.

Існують десятки систем управління контентом (CMS). Вибір неправильної платформи може коштувати дорого. Розглянемо найбільш привабливі варіанти CMS для організації електронної комерції.

Magento (рисунок 1.2). Широко використовуваний движок для інтернет-магазинів, пропонує безпрецедентну гнучкість, рівень контролю і функціонал. Є, мабуть, найпотужнішим фреймворком з відкритим вихідним кодом з усіх доступних. Сайт, створений на його основі, може бути легко розширений і модернізований.

Рис. 1.2. Magento CMS

Система пропонує досконале рішення для управління ресурсами, комфортна адмін-панель, і сотні варіантів конфігурації. Хоча розробка сайту на Magento і являє собою досить складний процес, згодом це воздасться великою гнучкістю і багатьма можливостями з управління сайтом після його запуску. Можливість вибору з широкого спектру його функцій, робить його ідеальним движком для різних категорій і галузей.

Хоча він має багато переваг, варто мати на увазі, що це багатство різноманітних функцій вимагає великої кількості ресурсів, щоб працювати безперебійно і гладко. Іншими словами, щоб магазин на Magento працював без збоїв, йому зазвичай необхідний хостинг з більш високою пропускною здатністю і більш потужним оснащенням сервера.

Magento адмін панель сайта (рисунок 1.3). Двіжок Magento є відмінним вибором, якщо вам потрібно додати складні пропозиції, купони, і валютні розрахунки. Список того, що може запропонувати Magento:

- налаштування варіантів доставки;
- численні варіанти оплати;
- можливість різних варіантів доставки;
- можливість динамічної зміни ціни;
- виконання замовлень;
- подарункові сертифікати;
- безпека;
- можливість додавання більшої функціональності;
- велика кількість форумів підтримки;
-

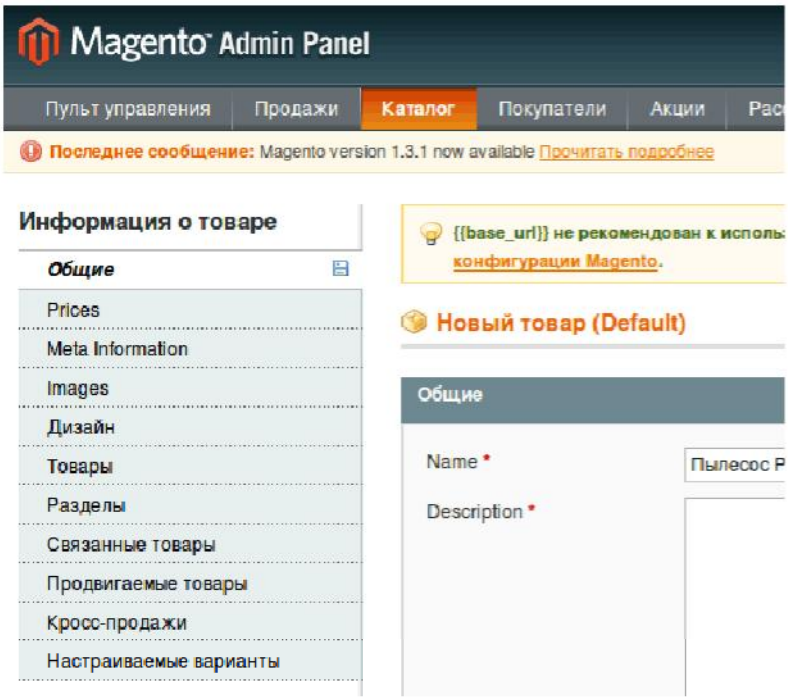


Рис. 1.3. Панель адміністрування Magento

1С Бітрікс (рисунок 1.4). Bitrix, безумовно, можна вважати однією з найкращих CMS, але - виключно для великих проектів з великими потребами, так як система вимагає потужних ресурсів, наприклад, виділеного сервера, і залучення професійних фахівців.

Рис. 1.4. 1С Бітрікс

На рисунку 1.5 представлено приклад управління сайтом в 1С Бітрікс

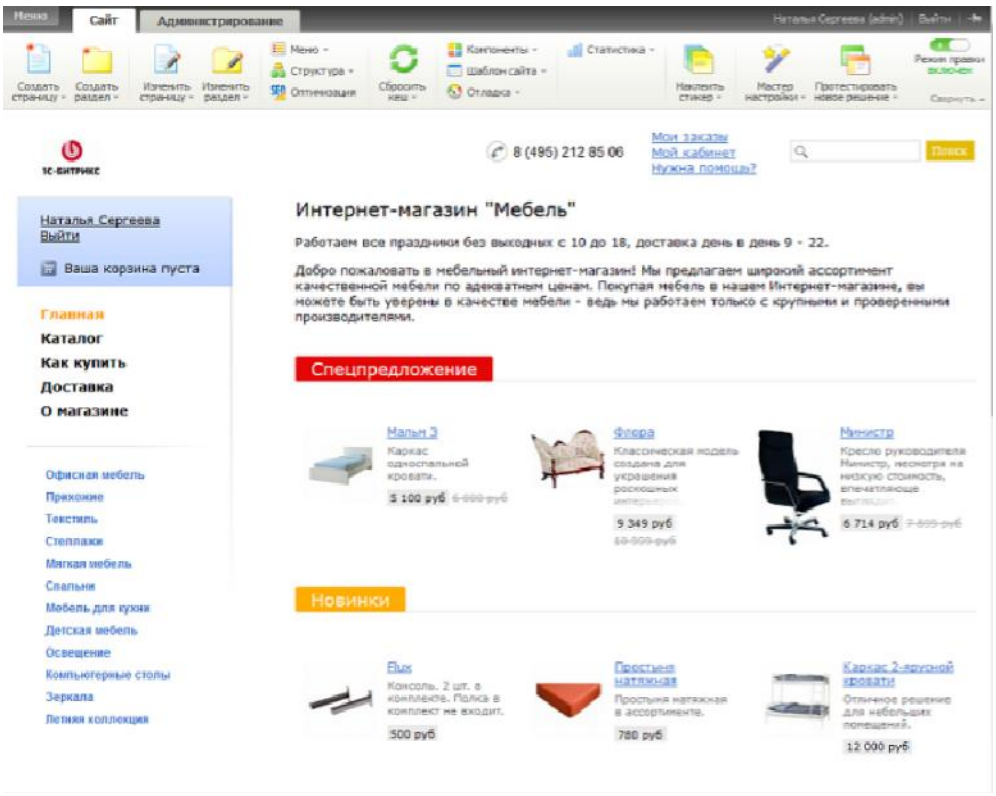


Рис. 1.5. Приклад реалізації сайту в 1С Бітрікс

OpenCart (рисунок 1.6). OpenCart є ще одним прикладом багатофункціонального движка, створеного для ведення електронної торгівлі. На відміну від Magento, він потрібує менше ресурсів, отже має і менше можливостей. Хоча OpenCart безкоштовний, використання його, як правило, вийде дорожче, якщо ви плануєте додавати нові функції і опції, так як більшість модулів доведеться купувати.

Платформа OpenCart вимагає набагато менше коштів, хоча в той же час надає всі необхідні функції, які можуть знадобитися для малого та середнього бізнесу. У порівнянні з Magento, OpenCart також має інтуїтивно зрозумілий інтерфейс.

Варто відзначити, що цей движок не підходить для кожного окремо взятого інтернет-магазину, але це відмінний варіант для простих, що будують плани і дивляться вперед, сайтів з самими звичайними кошиками для віртуальних покупок і системами замовлень.

Рис. 1.6. Система OpenCart

На рисунку 1.7 представлено зовнішній винляд адмін-панелі в системі OpenCart

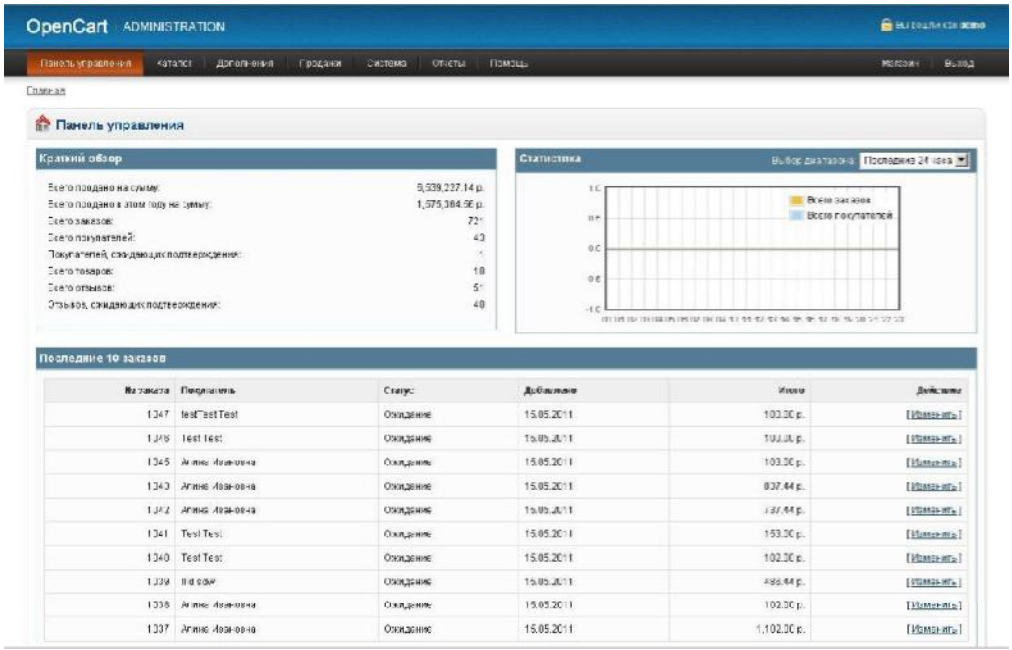


Рис. 1.7. Адмінка в системі OpenCart

Плагін WordPress для інтернет-магазинів (рисунок 1.8).Широкий спектр плагінів дає можливість додати весь функціонал, який потрібен для онлайн-продажів. При тому, що WordPress спочатку був створений для ведення блогів, на даний момент ситуація змінилася. Тепер же ми дійшли до того, що зараз існує більше повноцінних сайтів, побудованих на WordPress, ніж блогів.

Рис. 1.8. WordPress

Плагін WooCommerce (рисунок 1.9) в даний час є найбільш популярним для тих цілей, про які ми говоримо.

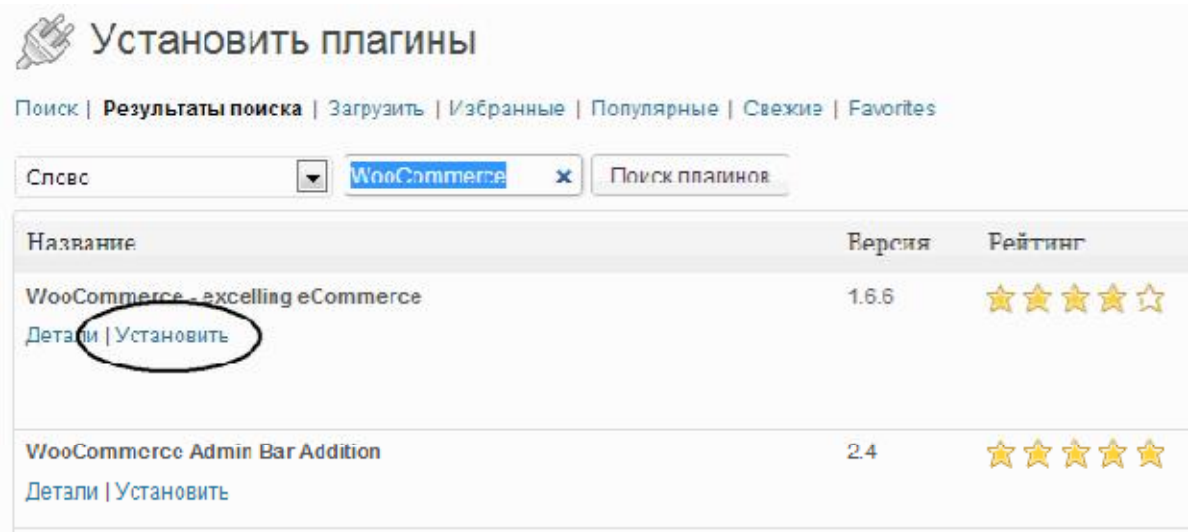


Рис. 1.9. Налаштування WooCommerce

Так само як і WordPress, Joomla (рисунок 1.10) є черговою (CMS) системою управління контентом з відкритим кодом. Спочатку вона була розроблена в якості платформи для візиток і корпоративних сайтів, а також для блогів. Кілька років тому розробники представили модуль, який пропонує основні функції для ведення електронної торгівлі.

Рис. 1.10. Joomla

Якщо ви збираєтеся розробляти інтернет-магазин лише з парою сотень товарів і з простими варіантами вибору модифікацій (наприклад колір, розмір тощо), ви, ймовірно, захочете ближче познайомитися з поєднанням Joomla + VirtueMart. Панель управління CMS (рисунок 1.11).

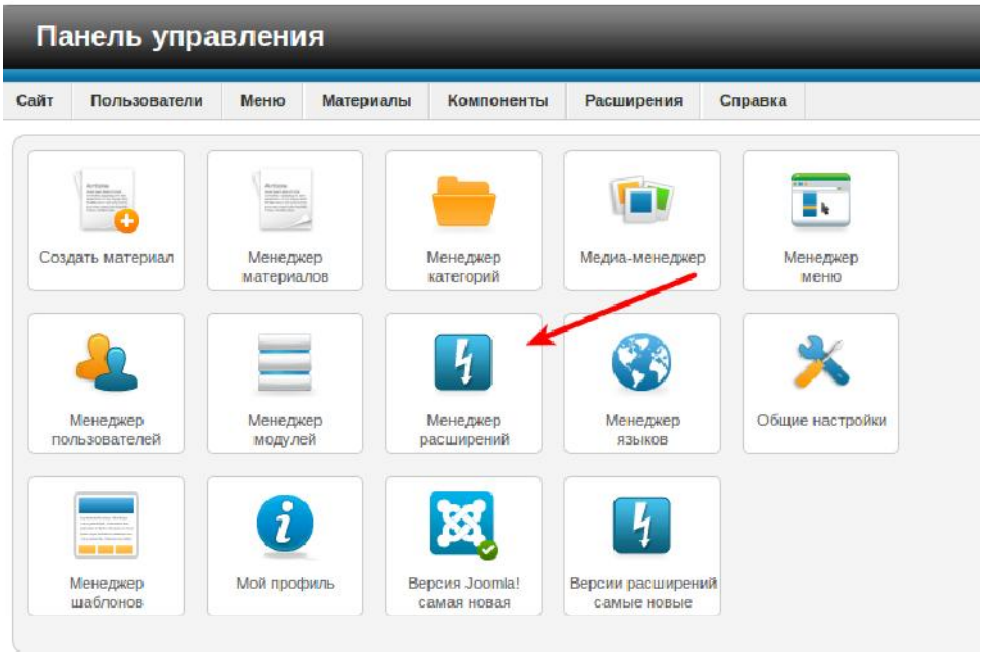


Рис. 1.11. Панель управління VirtueMart

Одним з головних плюсів є його швидкість роботи. VirtueMart, який, на мою думку, є найкращим модулем для Joomla, як правило, не працює зі стандартними шаблонами, і вам потрібно буде знайти або створити шаблон спеціально для цих потреб.

CS-Cart (рисунок 1.12). Використання CS-Cart дозволяє перебувати на плаву в даний час, коли з’явилася маса інтернет-магазинів. Відкрити магазин в Інтернеті стало так просто, як завести аккаунт електронної пошти на якому-небудь безкоштовному сервері. Особливістю CS-Cart стало те, що він перебував у постійному русі. Розробники інтегрували все нові і передові рішення, що, звичайно, дозволяло інтернет-магазинам, які використовують її, успішно конкурувати на світовому ринку.

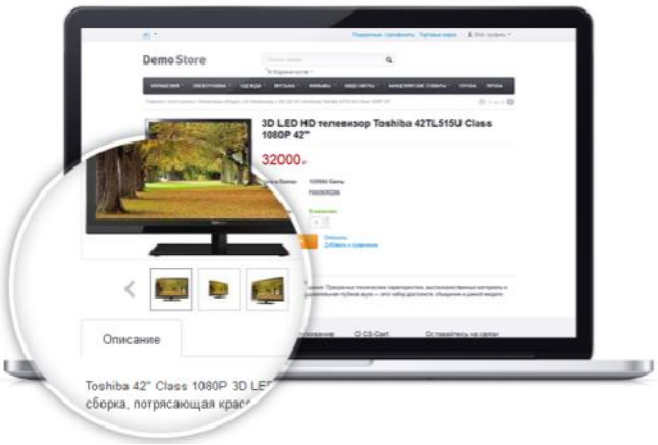


Рис. 1.12. CS-Cart

1.3. Специфікація вимог до системи

Перед тим, як приступати до розробки системи підтримки продажів, необхідно визначити які вирішуються з її допомогою завдання, її потенційних користувачів, а також роль системи серед інших додатків замовника: як існуючих, так і планованих.

Основним товаром, поширюваним компанією VDOM Box International, є web-сервер додатків VDOM Box, який об’єднує в собі апаратну і програмну частини. Варто відзначити, що, як правило, продається не сам сервер, а якесь «Готове рішення», в комплект поставки якого входить також якийсь набір пристроїв (наприклад, флеш-карта), додатків і послуг (наприклад, гарантійне обслуговування).

Первинними постачальниками товару є підрозділи компанії VDOM Box International. Від них товар потрапляє до дистриб'юторів, тобто оптових постачальників, які в свою чергу продають товар або іншим дистриб'юторам, або кінцевим споживачам.

Для управління продажами, перш за все, необхідно вести базу постачальників товару, тобто виробників (первинних постачальників), дистриб'юторів і кінцевих споживачів. Необхідно також відстежувати наявність товарів на складах кожного конкретного постачальника. Крім того існує необхідність реєструвати замовлення і відслідковувати їх виконання. До того ж, потрібно враховувати інформацію про те, скільки і яких найменувань товарів було замовлено і які конкретно товари були відправлені покупцеві, щоб таким чином відстежувати рух товару.

Система підтримки продажів під назвою VBI Partners, яка розробляється в роботі в першу чергу повинна вирішувати саме наведені вище завдання. Безпосередніми користувачами даної системи, що працюють з нею через web-інтерфейс, стануть первинні постачальники товарів і дистриб'ютори. Варто також відзначити, що перед компанією VDOM Box International стоять і інші завдання, суміжні з уже згаданими. За задумом компанії, подібні завдання повинні вирішуватися окремими додатками, які будуть інтегруватися з розроблюваною системою підтримки продажів VBI Partners.

Однією з таких задач є задача запису даних на смарт-карту. У комплект кожного «готового рішення» включається смарт-карта, на якій зберігається інформація про конфігурацію VDOM сервера (наприклад, його IP-адресу), а також про придбані користувачем ліцензії на ПЗ. За задумом замовника запис даних на смарт-карту повинен здійснюватися за допомогою спеціального пристрою і окремого desktop-додатку.

Другим завданням є проведення промо-акції, спрямованої на просування одного з програмних продуктів компанії VDOM Box International. Передбачається створення так званого web-сайту, що надає відповідні рекламні

матеріали, а також можливість зробити замовлення сервера VDOM Box в комплекті з ПЗ.

Сайт планується реалізувати у вигляді окремого web-додатки. Таким чином, система підтримки продажів буде з одного боку використовуватися через web-інтерфейс постачальниками товарів, а з іншого - виступати в якості ядра інтеграції з іншими додатками. Згідно з вимогами замовника, система підтримки продажів повинна вирішувати такі завдання:

- Ведення бази дистриб'юторів і клієнтів.
- Відстеження товару на складах.
- Реєстрація та проведення замовлень.
- Відстеження проходження товарів.
- Бути розроблена на платформі VDOM.
- Забезпечувати можливість інтеграції з іншими додатками.

На основі опису предметної області, зробленої замовником, можна виділити деякі ключові сутності. Розглянемо їх. Партнером називається особа (організація), яка бере участь в русі товару. Всіх партнерів можна умовно розділити на три категорії:

- Виробники. Первинні розповсюджувачі товарів. Поставляють товари дистриб'юторам.
- Дистриб'ютори. Закуповують товари у виробників і продають або іншим дистриб'юторам, або кінцевим споживачам.
- Кінцеві споживачі. Купують товари у дистриб'юторів для особистого користування.

Схема руху товарів представлена на рисунку 1.13.

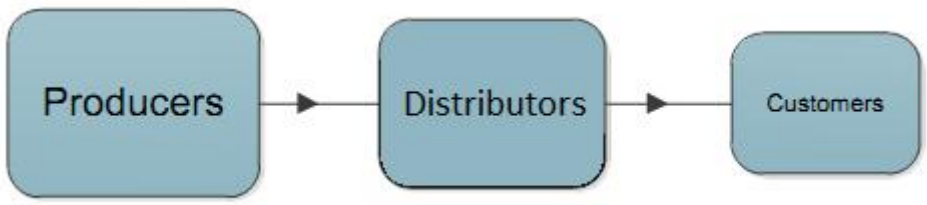


Рис. 1.13. Схема руху товарів компанії VDOM Box International

Партнери можуть складатися один з одним щодо «постачальник-споживач». Передбачається, що у кожного партнера є так звана «зона видимості», то є множина партнерів, з якими він працює.

Всі операції в системі, що розробляється підтримки продажів виконуються конкретними людьми - користувачами системи. Кожен користувач має логін і пароль для входу в систему. Після авторизації в системі він виконує всі свої дії від імені конкретного партнера.

У кожного партнера є довільне число складів, на яких розташовуються товари. Рух товарів відстежується саме між конкретними складами. Якщо партнер є приватною особою, то для нього має сенс виділити не склад, а наприклад, адресу місця доставки замовлення.

Щоб придбати якийсь товар партнер робить замовлення у постачальника. В заказі може фіксуватися наступна інформація:

- Вихідний склад (склад постачальника).
- Склад призначення (склад покупця).
- Позиції замовлення (найменування замовлених товарів і їх кількість).
- Конкретні товари, що доставляються на цільовий склад.
- Поточний стан замовлення.

За свого часу замовлення проходить ряд стадій, а саме:

- DRAFT (чорновий варіант). Замовлення завжди створюється в режимі DRAFT.

- CANCELLED (скасований). Покупець може з будь-якої причини відмовитися від замовлення. Скасовані замовлення поміщаються в архів.

- CONFIRMED (підтверджений). Покупця влаштовує замовлення і він його підтверджує.

- ASSIGNED (готовий до відправки). Зі складу відкладені всі необхідні для даного замовлення товари та замовлення готові до відправки.

- SENT (відправлений). Товари на замовлення відправлені.

- DELIVERED (доставлений). Товари отримані покупцем і переміщаються на цільовий склад. Замовлення виконано. Виконані замовлення поміщаються в архів.

Для кожного замовлення необхідно відстежувати його поточний стан, а також вести історію зміни цих станів.

Всі товари, поширювані VDOM Box International можна умовно розділити на три групи: апаратна частина, програмне забезпечення, послуги, а також «готові рішення ». Перші три категорії представляють так звані прості товари, четверта ж - складні, тобто що складаються з певного числа простих. Наприклад, готове рішення VDOM Box EOF Edition включає в себе апаратну частину: сервер VDOM Box, флеш-карту, смарт-карту; ПЗ: додаток Eye on Files Online; додаткові послуги: установку, гарантійне обслуговування.

Одиниця товару може бути як простий (наприклад, конкретна флеш-карта), так і складовою (конкретна система VDOM Box EOF Edition). Певний товар (послуга) характеризується набором атрибутів і їх значень. Отже, у розглянутій предметній області можна виділити наступні ключові сутності:

- Партнер.
- Користувач системи.
- Склад.
- Замовлення.
- Найменування товару.
- Одиниця товару.

Маючи в своєму розпорядженні загальну концепцію VBI Partners, список вимог до неї і набір ключових сутностей предметної області, можна переходити безпосередньо до розробки системи.

В об’єктній моделі виділяється два основні класи об’єктів: об’єкти-контейнери і прості об’єкти. Основне завдання контейнера - задавати логічну структуру сторінку web-додатку, а також визначати тип виведення вмісту.

Передбачається реалізація наступних типів контейнерів:

- HTML container: відповідає за виведення об’єктів в HTML-потік.

- Flash container: відповідає за виведення об’єктів у flash-анімацію.
- PDF container.
- PNG container.

Також за допомогою певних типів контейнерів можна створювати складові об’єкти, такі як «таблиця». Для цих цілей запроваджено такі типи контейнерів:

- Table container: відповідає за виведення таблиці повністю, містить набір об’єктів типу row-container.
- Row container: відповідає за відображення одного ряду таблиці.
- Cell container: відповідає за відображення однієї комірки таблиці.

Такий поділ типів дозволяє дуже гнучко конструювати сторінки web-додатку. На рисунку 1.14 показаний приклад структури сторінки: верхній контейнер це HTML-container, який містить кілька об’єктів усередині себе: два інших контейнера (Flash container, PNG container) і п’ять простих об’єктів (два Picture object, Menu object, два RichText Object).

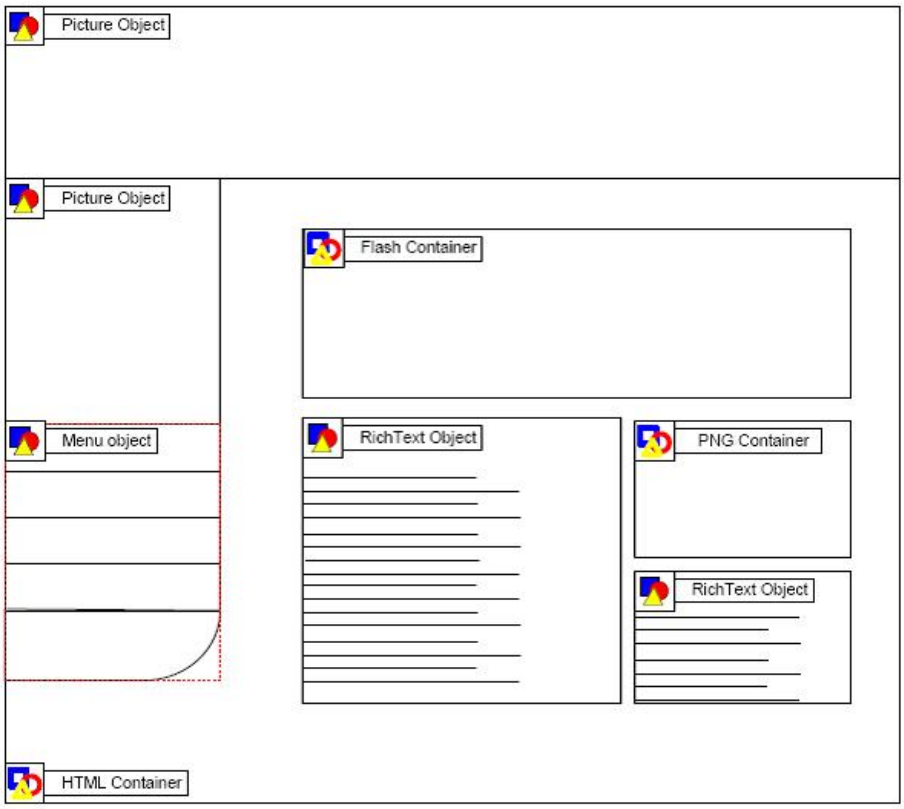


Рис. 1.14. Приклад структури сторінки web-додатке з використанням об’єктів різних типів

Для представлення об’єктів в системі використовується XML-формат опису типів. Це дозволяє легко створювати призначені для користувача об’єкти і переносити їх з однієї системи в іншу. Також таке представлення даних забезпечує потрібну динамічність для роботи сервера. Опис типу має наступну структуру:

```
<Type>
        <Information>...</Information>
        <Attributes>...</Attributes>
        <Resources>...</Resources>
        <SourceCode>...</SourceCode>
</Type>
```

Секція Information описує загальну інформацію типу: ім’я, опис, ідентифікатор, типи підтримуваних контейнерів. Секція Attributes містить в собі опис атрибутів об’єктів даного типу і використовується при трансляції XML-опису у внутрішнє представлення. Секція Resources описує різні файлові ресурси, використовувані типом, наприклад, це можуть бути картинки або іконки. Секція SourceCode містить в собі код на мові Python, який реалізує функціональність даного класу. Тут міститися функції щодо виведення представлення об’єкта в різні типи контейнерів і додаткові функції, які об’єкт надає.

Даний XML-документ обробляється на етапі завантаження сервера, і в результаті формується внутрішнє представлення об’єкта в пам’яті, яке використовується при формуванні об’єктів web-додатку.

Специфікація вимог до програмної системи – це специфікація окремого програмного продукту, програми або набору програм, які виконують деякі дії в деякому середовищі. Тобто – це повний опис поведінки системи що розробляється [3].

В загальному випадку специфікація включає наступне:

- глосарій проекту;
- опис варіантів використання.

Наведемо список основних термінів та понять в області розробки програмної системи реалізації товарів – глосарій. Глосарій – список понять в

специфічній області знання з їх визначеннями [3]. Ці поняття та визначення подано у таблиці 1.1.

Таблиця 1.1

Глосарій

Термін	Опис терміну
1. Основні поняття та категорії предметної області та проекту	
Товар	Продукт природи і людської праці або тільки людської праці у матеріальній і нематеріальній субстанції та у формі послуг, який завдяки своїм властивостям здатен задовольняти наявні чи передбачувані суспільні потреби і призначений для обміну і купівлі-продажу; продукт праці, що виробляється не для власного споживання, а на продаж, а також матеріальні та нематеріальні активи, а також цінні папери та деривативи, що використовуються у будь-яких операціях, крім операцій з їх випуску (емісії) та погашення.
Каталог товарів	Перелік товарів з їх докладним описом
Категорії товарів	Сортування товарів за певної класифікаційною ознакою
2. Користувачі системи	
Оператор	Користувач системи, який може переглядати товари, переглядати, додавати та редагувати категорії та каталоги, опрацьовує замовлення, здійснює супровід замовлень
Постачальник	Особа, яка здійснює постачання товарів у відповідності до замовлень
Адміністратор	Користувач системи, який, окрім виконання функцій користувача, може здійснювати управління користувачами системи
Клієнт	Особа, яка здійснює операції, які пов’язані з купівлею товарів
Курєр	Особа, яка здійснює доставку товарів клієнту

1. Вхідні та вихідні документи	
Перелік товарів	Список товарів із функціональним описом
Перелік категорій товарів	Даний документ необхідний для виведення переліку категорій товарів
Замовлення товару	Документ, що представляє собою опис процесу купівлі товарів
Документ про стан оплати та доставки товару	Документ, що представляє собою опис процесів, що стосуються оплати та доставки товару

На рисунку 1.15 представлена діаграма прецендентів системи реалізації товарів, яка буде реалізована з використанням технології VDOM.

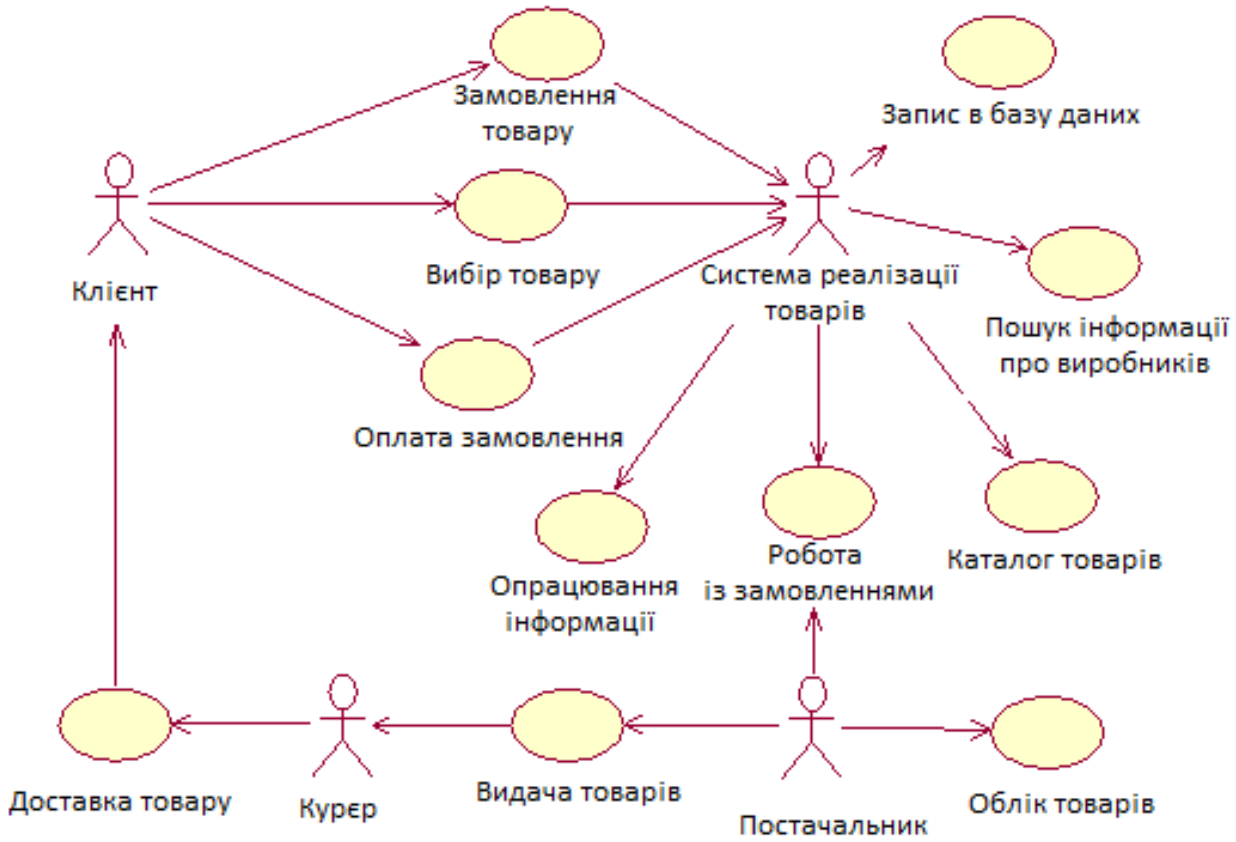


Рис. 1.15. Діаграма варіантів використання системи реалізації товарів

У таблицях 1.2 – 1.11 наведено опис варіантів використання, що реалізують основну функціональність програмної системи.

Таблиця 1.2

Варіант використання «Вибір товару»

Контекст використання	Формування замовлень.
Дійові особи	Клієнт
Передумова	Користувач аутентифікований та авторизований.
Тригер	Натиснення кнопки «Вибрати товар».
Сценарій	1. Заповнення полів. 2. Перевірка правильності введених даних. 3. Натиснення кнопки «Зберегти».
Постумова	Відображення інформації про вибраний товар.

Таблиця 1.3

Варіант використання «Замовлення товару»

Контекст використання	Формування замовлень.
Дійові особи	Клієнт
Передумова	Користувач аутентифікований та авторизований.
Тригер	Натиснення кнопки «Замовити товар».
Сценарій	1. Заповнення полів. 2. Перевірка правильності введених даних. 3. Натиснення кнопки «Зберегти».
Постумова	Відображення інформації про статус замовленого товару.

Таблиця 1.4

Варіант використання «Оплата замовлення»

Контекст використання	Управління замовленнями.
Дійові особи	Клієнт
Передумова	Користувач аутентифікований та авторизований.
Тригер	Натиснення кнопки «Оплатити замовлення».
Сценарій	1. Заповнення полів. 2. Перевірка правильності введених даних. 3. Натиснення кнопки «Зберегти».
Постумова	Відображення інформації про статус оплати замовленого товару.

Таблиця 1.5

Варіант використання «Робота із замовленнями»

Контекст використання	Управління замовленнями.
Дійові особи	Постачальник
Передумова	Користувач аутентифікований та авторизований.
Тригер	Натиснення кнопки «Переглянути замовлення».
Сценарій	1. Заповнення полів. 2. Перевірка правильності введених даних. 3. Натиснення кнопки «Зберегти».
Постумова	Відображення інформації про історію замовлень

Таблиця 1.6

Варіант використання «Облік товарів»

Контекст використання	Управління товарами.
Дійові особи	Постачальник
Передумова	Користувач аутентифікований та авторизований.
Тригер	Натиснення кнопки «Облік товарів».
Сценарій	1. Заповнення полів. 2. Перевірка правильності введених даних. 3. Натиснення кнопки «Зберегти».
Постумова	Відображення інформації про облік товарів

Таблиця 1.7

Варіант використання «Видача товарів»

Контекст використання	Управління товарами.
Дійові особи	Постачальник
Передумова	Користувач аутентифікований та авторизований.
Тригер	Натиснення кнопки «Рух товарів».
Сценарій	1. Заповнення полів. 2. Перевірка правильності введених даних. 3. Натиснення кнопки «Зберегти».
Постумова	Відображення інформації про статус товару

Таблиця 1.8

Варіант використання «Доставка товару»

Контекст використання	Управління товарами.
Дійові особи	Курєр
Передумова	Користувач аутентифікований та авторизований.
Тригер	Натиснення кнопки «Доставка товару».
Сценарій	1. Заповнення полів. 2. Перевірка правильності введених даних. 3. Натиснення кнопки «Зберегти».
Постумова	Відображення інформації про доставку товару

Таблиця 1.9

Варіант використання «Запис в базу»

Контекст використання	Управління товарами.
Дійові особи	Оператор
Передумова	Користувач аутентифікований та авторизований.
Тригер	-
Сценарій	1. Заповнення полів. 2. Перевірка правильності введених даних. 3. Натиснення кнопки «Зберегти».
Постумова	Відображення інформації товари в базі даних

Таблиця 1.10

Варіант використання «Каталог товарів»

Контекст використання	Управління товарами.
Дійові особи	Оператор
Передумова	Користувач аутентифікований та авторизований.
Тригер	Формування каталогу товарів
Сценарій	1. Заповнення полів. 2. Перевірка правильності введених даних. 3. Натиснення кнопки «Зберегти».
Постумова	Відображення каталогів товарів

Таблиця 1.11

Варіант використання «Пошук інформації про виробників»

Контекст використання	Управління постачальниками.
Дійові особи	Оператор
Передумова	Користувач аутентифікований та авторизований.
Тригер	Формування каталогу товарів
Сценарій	1. Заповнення полів. 2. Перевірка правильності введених даних. 3. Натиснення кнопки «Пошук».
Постумова	Відображення результатів пошуку

За результатами створеної розкадровки було сформульовано набір функціональних та нефункціональних вимог до системи, які представлені в таблиці 1.12 та таблиці 1.13 відповідно.

Таблиця 1.12

Функціональні вимоги

Ідентифікатор вимоги	Назва вимоги (варіанту використання)	Атрибут вимоги		
		Пріоритет	Складність	Контакт
1	Вибір товару	Обов’язкова	Середня	Клієнт
2	Замовлення товару	Обов’язкова	Висока	Клієнт
3	Оплата замовлення	Обов’язкова	Висока	Клієнт
4	Робота із замовленнями	Обов’язкова	Низька	Постачальник
5	Облік товарів	Обов’язкова	Середня	Постачальник
6	Видача товарів	Обов’язкова	Висока	Постачальник
7	Доставка товару	Обов’язкова	Середня	Курєр
8	Запис в базу	Обов’язкова	Середня	Оператор
9	Каталог товарів	Обов’язкова	Висока	Оператор
10	Пошук інформації про виробників	Обов’язкова	Середня	Оператор

Специфікацію нефункціональних вимог наведено в таблиці 1.13.

Наведемо специфікацію суттєвих для проекту нефункціональних вимог:

1. Застосовність:
   - мінімальний час для навчання звичайних і досвідчених користувачів;
   - відповідність стандартам графічного інтерфейсу.
2. Надійність:
   - постійна безвідмовна робота;
   - пропускна здатність каналу зв’язку 100 Mb/s;
   - забезпечення можливості віддаленого доступу до комп’ютера, на якому буде встановлена система;
   - доступність – 5%.
3. Робочі характеристики:
   - швидкість завантаження інтернет-ресурсу: 0,1 – 1 с;
   - число транзакцій: 100 / 1 с;
   - використання ресурсів: від 1 Gb, в залежності від кількості користувачів.
4. Проектні обмеження:

- Операційна система FreeBSD;
- VDOM BOX Server;
- СУБД Embedded SQLite 3
- Python;

Таблиця 1.13

Специфікація нефункціональних вимог

Ідентифікатор вимоги	Назва вимоги	Атрибути вимог		
		Пріоритет	Складність	Контакт
Застосовність				
1.1	Час, необхідний для навчання звичайних і досвідчених користувачів	Рекомендована	Низька	Адміністратор
1.2	Основні вимоги застосовності нової системи відносно інших систем, які знають користувачі	Опційна	Низька	Адміністратор
1.3	Вимоги по відповідальності стандартам графічного інтерфейсу користувача	Рекомендована	Низька	Адміністратор
Надійність				
2.1	Доступність	Обов’язкова	Середня	Адміністратор
2.2	Середній час безвідмовної роботи	Рекомендована	Середня	Адміністратор
2.3	Точність	Обов’язкова	Середня	Адміністратор
Робочі характеристики				
3.1	Використання ресурсів	Рекомендована	Середня	Адміністратор
4.1	Вимоги до технології програмування	Рекомендована	Середня	Адміністратор

5. Вимоги до документації
   - наявність інтерактивної довідки.
6. Інтерфейси:

   -інтерфейс користувача – Web- інтерфейс.

Висновки до розділу 1

Здійснено опис предметної області, напрями діяльності. Визначено склад функцій, що входять до бізнес-процесу на основі яких розроблено схему управління бізнес-процесом. Проведено аналіз відомих програмних систем. Здійснено аналіз вимог до програмної системи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИСТЕМИ РЕАЛІЗАЦІЇ ТОВАРІВ НА ОСНОВІ VDOM

2.1. Розроблення архітектури програмної системи

Проектування додатку в цілому зводиться до проектування його окремих частин. На основі інформації про системе підтримки продаж, отриманої від замовника, а також вивчення цільової платформи було вирішено, що система буде складатися з наступних частин: реляційної бази даних, шару бізнес-логіки, зовнішнього API і web-інтерфейсу. Опис процесу проектування кожної з згаданих частин розглядається у відповідних підрозділах даної роботи.

Розглянемо архітектуру системи на базі VDOM (рисунок 2.1).

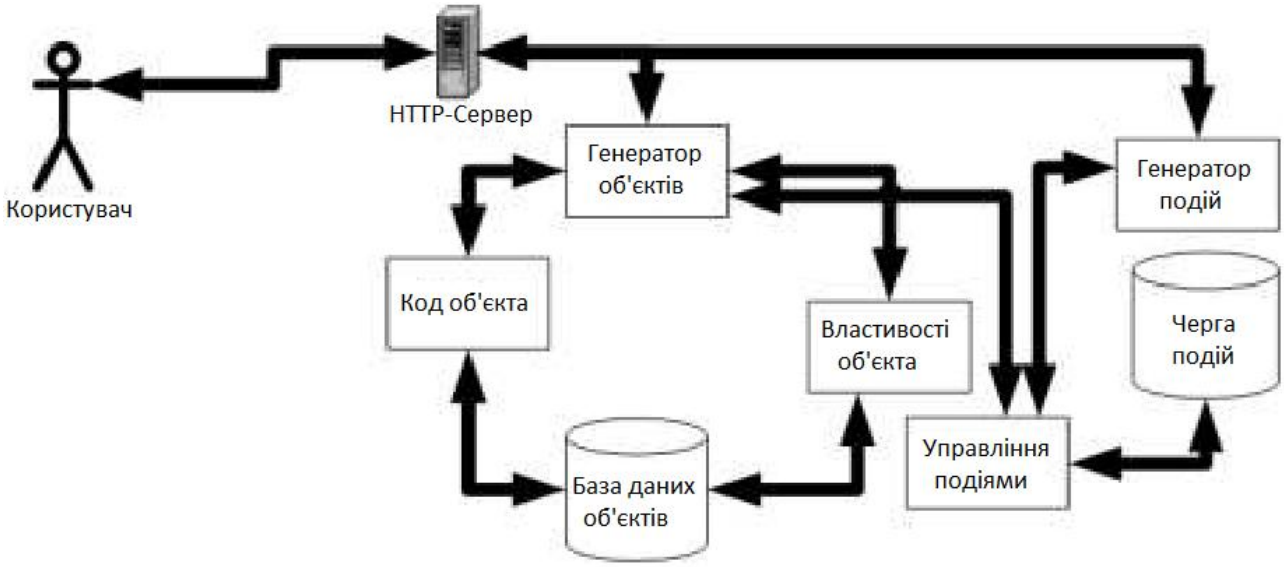


Рис. 2.1. Архітектура ситеми на базі VDOM

Основний елемент системи - база даних об’єктів, яка містить всю необхідну для роботи системи інформацію. У базі для кожного об’єкта додатку міститься два типи записів: запис про властивості об’єкта і запис про код об’єкта. На основі цієї інформації генератор об’єктів створює екземпляр об’єкта з заданими властивостями і функціональністю.

Інша важлива підсистема - управління подіями. Завдяки їй можна робити "асинхронний web-інтерфейс", що актуально для web-додатків з високим рівнем інтерактивності і наявністю зворотнього зв’язку.

На рисунку 2.2 представлено схему завантаження web-додатку в VDOM_Memory.

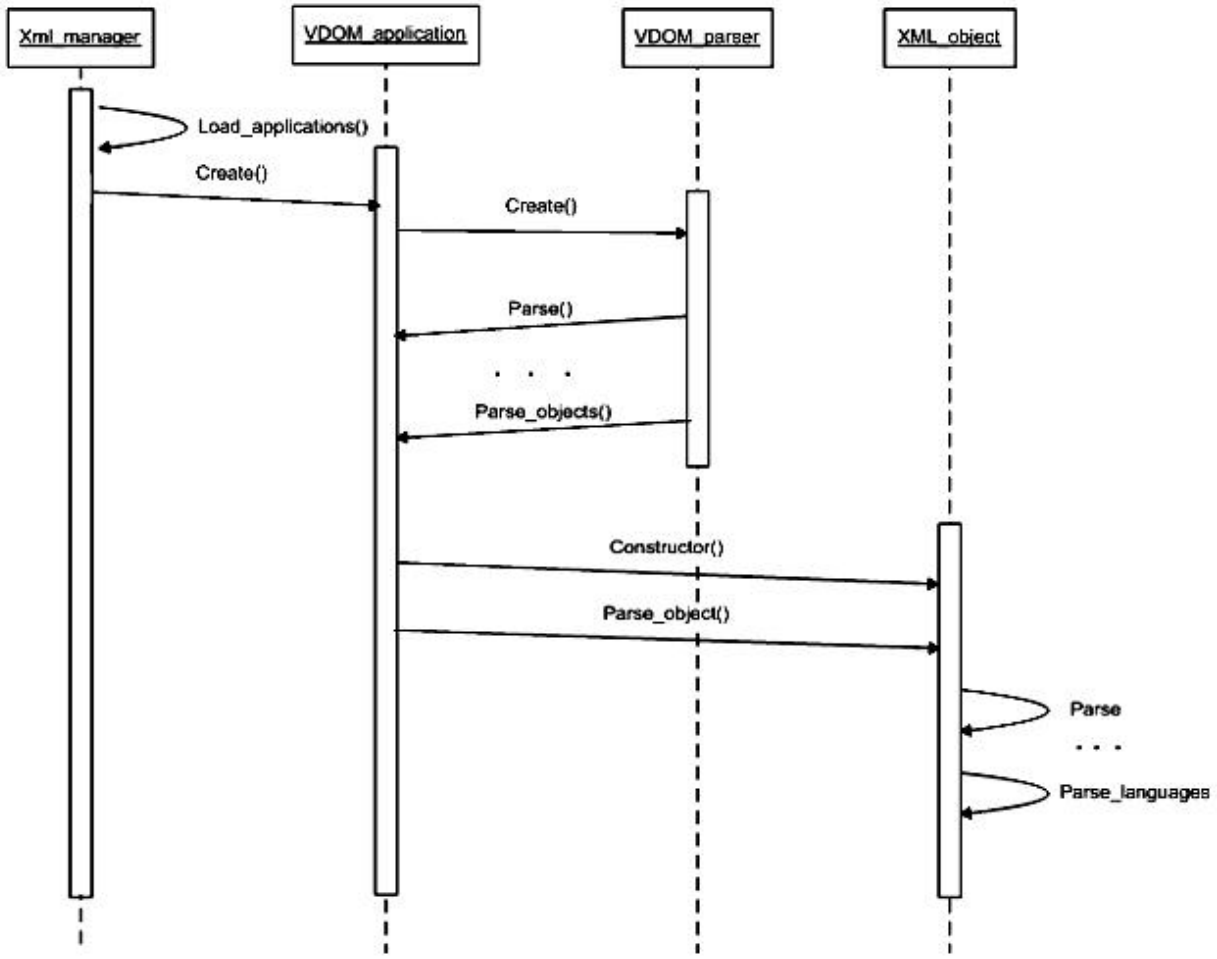


Рис. 2.2. Діаграма послідовності завантаження сервера та додатку у VDOM_Memory

В даному підрозділі розглядається проектування архітектури системи VBI Partners, що складається з шару бізнес-логіки і зовнішнього API, необхідного для інтеграції з іншими додатками.

Шар бізнес-логіки. При проектуванні системи підтримки продажів VBI Partners, було прийнято рішення розробити окремий шар бізнес-логіки, тобто деякої ланки між базою даних і іншою частиною програми. Таке рішення саме

по собі є вдалим в архітектурному плані. Крім того за рахунок виділення бізнес-логіки в окремий шар вдалося вирішити ряд важливий завдань, а саме:

- Зниження дублювання коду. Очевидно, що в різних частинах програми (в першу чергу на різних web-сторінках і в операціях зовнішнього API) будуть використовуватися подібні операції доступу до бази даних. Введення шару бізнес-логіки дозволяє зібрати типові операції звернення до БД в одному місці і, таким чином, істотно знизити дублювання коду.

- Зниження залежності додатку від використовуваної СУБД. У міру того як кількість користувачів і обсяги даних системи будуть зростати, може наступити момент, коли вбудована база даних перестане справлятися з навантаженням, і знадобиться переводити систему на повноцінну СУБД (наприклад, MySQL). У цьому випадку досить буде змінити в самому додатку тільки шар бізнес-логіки, та й то лише частково, тобто з урахуванням особливостей SQL-діалекту нової СУБД. Решта ж частини програми при цьому міняти не доведеться.

Реалізація обмежень цілісності. Як вже було сказано раніше, СУБД системи VDOM не підтримує завдання специфічних для предметної області обмежень цілісності через відсутність підтримки тригерів. Однак дану задачу вдалося повністю вирішити на рівні шару-бізнес логіки.

Першим кроком проектування був вибір способу організації бізнес-логіки. Після аналізу існуючих підходів, описаних в [3], був зроблений вибір на користь «Моделі предметної області» (Domain Model), так як наша предметна область є складною і, можливо, буде ускладнюватися в майбутньому. Обраний підхід передбачає, що класи бізнес-логіки та їх асоціації в певному сенсі відповідають сутностям предметної області і відношень між ними.

Другим кроком проектування був вибір способу виконання перетворень ORM (Object-Relational Mapping) [4], тобто перетворення даних з реляційного подання до об’єктного і навпаки. З описаних в [3] варіантів був обраний спосіб, відомий під назвою Active Record, так як він є відносно простим в реалізації і добре поєднується з організацією бізнес-логіки за принципом моделі

предметної області. Патерн Active Record має на увазі, що кожен клас містить у собі як бізнес-логіку, так і методи доступу до бази даних (вибірка, створення, оновлення, видалення).

Третім кроком проектування стало побудова для шару бізнес-логіки діаграми класів в нотації UML [5]. При вирішенні даного завдання враховувалися опис предметної області, схема розробленої бази даних, а також рішення прийняті на двох попередніх етапах проектування.

Розглянемо тепер структуру типового класу шару бізнес-логіки. Найчастіше одному класу відповідає одна таблиця в БД. Деяким класам, однак, відповідають кілька таблиць. В цьому випадку вибірка і збереження об'єкта такого класу зачіпає всі відповідні йому таблиці бази даних.

Типовий клас шару бізнес-логіки містить:

- Поля відповідних йому в БД таблиць (в тому числі сурогатний первинний ключ головної таблиці, а також зовнішні ключі) і, можливо, деякі додаткові атрибути. Доступ до всіх полів класу, за винятком зовнішніх ключів, здійснюється безпосередньо, тобто без використання access-методів. Зовнішні ключі є закритими атрибутами класу.

- Статичний метод fill_from_row(row), який повертає persistent об'єкт (тобто об'єкт, для якого є відповідність в базі даних), по вектору row, який представляє собою рядок вибірки з БД. Даний метод застосовується як допоміжний всередині методів вибірки об'єктів даного класу.

- Статичні методи вибірки об'єктів даного класу. Імена таких методів починаються, як правило, з префікса «get_by». До цієї категорії входять такі методи, як наприклад, get_by_id (id) (вибрати об'єкт за значенням первинного ключа) і get_all () (вибрати всі об'єкти даного класу). Особливу категорію складають методи для вибірки об'єктів даного класу за критерієм зв'язку з об'єктами інших класів, наприклад, метод User.get_by_partner (partner) (Отримати список всіх користувачів, які належать даному партнеру). Такого роду методи є закритими і не призначені для виклику клієнтом шару бізнес-логіки (будемо називати їх «закритими getter-ами»). Передбачається, що вони

будуть використовуватися всередині методів інших класів. Наприклад, згаданий вище метод User.get_by_partner (partner) буде використовуватися всередині методу get_users() класу Partner. Зауважимо, що в цьому випадку метод get_users () не буде безпосередньо звертатися до бази даних, і відповідно не буде містити всередині себе SQL-запити. Такий підхід обумовлений прагненням зібрати всі SQL-запити до відповідної таблиці (таблиць) всередині одного класу, щоб при наступних змінах структури БД або переході на іншу СУБД знайти і змінити дані SQL-запити було просто. Досить важливим є питання про те, як технічно зробити закриті getter-и доступними виключно для класів шару бізнес-логіки. У деяких мовах програмування існують засоби для вирішення даного завдання, наприклад механізм дружніх класів в C++. Однак в цілому підсумкова реалізація тут сильно залежить від використовуваної мови програмування.

- Методи синхронізації з базою даних. Зауважимо, що зміна атрибутів об’єктів, а також виклик більшості його методів не змінюють об’єкт в БД. Синхронізація з базою даних відбувається в момент виклику методу save(). По перше, даний метод виконує збереження поточного стану об’єкта в БД. Якщо поточний об’єкт є non-persistent, тобто для нього ще немає відповідності в БД (такий об’єкт може бути створений за допомогою виклику конструктора данного класу), то всередині методу save() для нього викликається метод insert() (створення нового об’єкта в БД). Для persistent ж об’єкта викликається метод update() (Оновлення поточного об’єкта в БД). По-друге, саме всередині методу save() виконується перевірка всіх обмежень цілісності, які можуть порушитися при зміні даного об’єкта. Зауважимо, що в жодного з класів немає звичного методу delete(), призначеного для видалення об’єкта з БД. Це пояснюється тим, що в системі, що розробляється необхідно зберігати повну історію переміщення товарів. Інформація про переміщення в свою чергу охоплює практично всі класи шару бізнес-логіки, тому для підтримки цілісності історії видаляти будь-які об’єкти в системі заборонено.

- Методи для вибірки об’єктів, пов’язаних з об’єктом даного класу. Прикладом такого методу є метод get_users() класу Partner видає список всіх користувачів, які належать даному партнеру. Як вже було сказано, реалізація цих методів припускає використання закритих getter-ів відповідних класів.

- Методи, які виконують зв’язування об’єкта з об’єктами інших класів (для відношень типу «один-до-багатьох»). Зв’язок, створюваний даними методами, є тимчасовим, тобто пов’язує саме об’єкти бізнес-логіки без звернення до БД. Фіксування ж встановленого зв’язку в базі даних виконується при виклику методу save() підлеглого об’єкта. Наприклад, для призначення користувача user партнеру partner необхідно спочатку викликати метод partner.add_user (user), а потім зберегти підлеглий об’єкт за допомогою виклику user.save(). Зауважимо, що зв’язування об’єктів відповідно до ставленням типу «багато-до-багатьох» виконується по-своєму в кожному окремо взятому випадку.

- Методи, що реалізують бізнес-логіку. Результатом проведеної роботи з проектування шару бізнес-логіки стала UML діаграма класів, ознайомитися з якою можна на рисунку 2.3, а також в додатку.

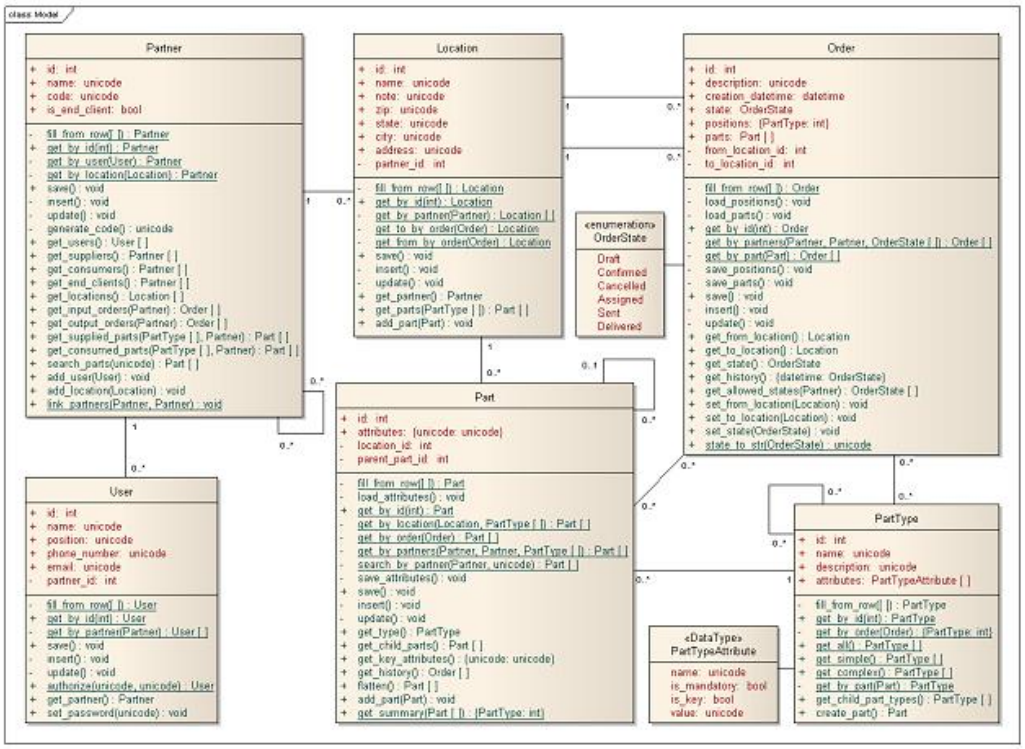


Рис. 2.3. Діаграма класів шару бізнес-логіки

Для інтеграції розроблюваної системи з іншими додатками, необхідно було розробити для неї зовнішній API. Згідно з вимогою замовника такої API повинен являти собою якийсь набір окремих операцій, які приймають і відправляють дані в форматі XML. Також замовником була надана детальна специфікація даного API. Наведемо короткий список операцій, що становить зовнішній API:

- login - авторизувати користувача в системі.
- create_end_client - створити клієнта зазначеного партнера.
- create_user - створити користувача вказаного партнеру.
- create_location - створити склад вказаного партнеру.
- create_order - створити замовлення.
- get_locations - отримати список складів, що належать зазначеному партнеру.
- get_part_types - отримати список всіх найменувань товару.

2.2. Проектування структури бази даних

В якості вирішення задачі зберігання даних була обрана розробка реляційної БД. Завдання проектування БД зводиться до побудови її схеми.

Рішення почати проектування БД з побудови ER-схеми було прийнято в силу більшої виразності ER-моделі даних в порівнянні з реляційною. При вирішенні даного завдання була використана ER-нотація Чена [1]. На основі опису предметної області та списку вимог до системи, була побудована ER-схема, діаграма якої представлена на рисунку 2.4, а також в додатку. У таблицях 2.1 і 2.2 наведено докладний опис відповідно множин сутностей і множин зв’язків даної схеми.

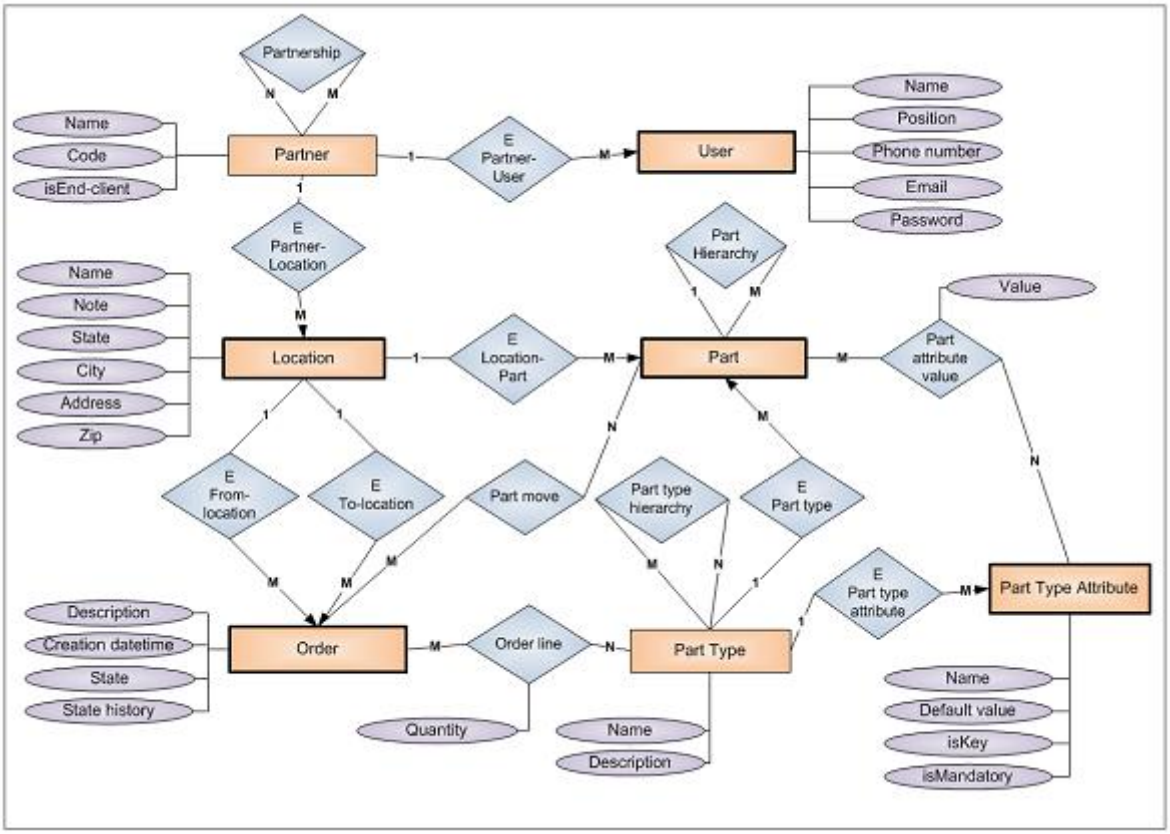


Рис. 2.4. ER-діаграма

Таблиця 2.1

Опис множин сутностей ER-схеми

Назва атрибутів	Тип атрибутів	Опис атрибутів
Part	Одиниця товару	
Part Type	Найменування товару	
Description	Стрічка	Опис найменування товару.
Name	Стрічка	Назва найменування товару.
Part Type Attribute	Атрибут товарів певного найменування	
Name	Стрічка	Назва атрибуту.
Default value	Стрічка	Значення атрибута за замовчуванням.
isKey	Логічний	Перемикач, який визначає, чи є атрибут «Ключовим» (тобто особливо важливих при описі товару
isMandatory	Логічний	Перемикач, який визначає, чи є атрибут обов’язковим
Partner	Партнер (особа, організація), що бере участь в русі товарів	
Name	Стрічка	Назва організації / ім’я кінцевого клієнта.
Code	Стрічка	Спеціальний код партнера (використовується для швидкої ідентифікації партнера, а також для відмінності партнерів з однаковими іменами).

isEnd-client	Логічний	Перемикач, який визначає, чи є партнер кінцевим споживачем чи ні.
User	Користувач системи VB Partners	
Name	Стрічка	Ім’я користувача.
Position	Стрічка	Посада, яку займає користувачем в його компанії.
Phone number	Стрічка	Номер телефону, за яким можна зв’язатися з користувачем.
Email	Стрічка	Адреса електронної пошти користувача. Також використовується в якості логіна для авторизації в системі.
Password	Стрічка	Пароль користувача для авторизації в системі.
Location	Склад (інший пункт відправлення / призначення замовлення)	
Address	Стрічка	Адреса (вулиця, будинок) розташування складу.
State	Стрічка	Район, в якому знаходиться склад.
City	Стрічка	Місто, в якому знаходиться склад.
Note	Стрічка	Опис складу.
Name	Стрічка	Назва складу.
Zip	Стрічка	Поштовий індекс складу.
Order	Замовлення	
Creation date	Дата, час	Дата створення замовлення.
State	Стан замовлення (draft, confirmed, cancelled, assigned, sent)	Поточний стан замовлення.
Description	Стрічка	Опис замовлення.
State history	Дата, час, стан	Історія зміни стану замовлення. Являє собою список значень типу <Дата, час; стан>.

Таблиця 2.2

Опис множин зв’язків ER-схеми

Назва	Множини сутностей	Тип відношення	Атрибути	Семантика
Partner-User	Partner, User	1:Б		Даний користувач належить данному партнеру і відповідно може працювати в системі від його імені. Партнер може мати кілька користувачів. Користувач може належати одному і тільки одному партнеру.
Partnership	Partner,	Б:N		Дані партнери складаються в відносинах партнерства, один з них є постачальником, інший - споживачем. Партнер може мати довільне

				число постачальників і споживачів. Один партнер не може бути одночасно постачальником і споживачем для іншого партнера.
Partner-Location	Partner, Location	1:Б		Даний склад належить даному партнеру. Партнер може мати довільне число складів. Склад може належати одному і тільки одному партнеру.
From- location	Location, Order	1:Б		Даний склад є пунктом відправки даного замовлення. З одного складу може бути відправлено (Підготовлено до відправки) довільну кількість замовлень. Замовлення може бути відправлене з одного і тільки одного пункту.
To-location	Location, Order	1:Б		Даний склад є пунктом призначення даного замовлення. На один склад може бути відправлено (Підготовлено до відправки) довільне число замовлень. Замовлення може бути відправлено на один і тільки на один склад.
Part hierarchy	Part, Part	1:Б		Один товар є складовою частиною іншого товару. Товар може складатися з довільного числа інших товарів. Товар може бути складовою частиною максимум одного товару.
Location-Part	Location, Part	1:Б		Даний товар знаходиться на даному складі, або відправлений у відповідності з замовленням з даного складу на інший. На одному складі може перебувати довільне число товарів. Один товар може знаходитися на одному і тільки одному складі.
Part move	Part, Order	Б:N		Даний товар відправлений відповідно до цього замовленням. Товар може бути відправлений в різний час з довільним числом замовлень. Відповідно до конкретного замовленням може бути відправлено довільне число товарів.
Part type	Part Type, Part	1:Б		Даний товар є товаром даного найменування. Одне найменування можуть мати різні товари. Товар може мати одне і тільки одне найменування.
Order line	Order, Part	Б:N	Quantity -	В даному замовленні вказано

	Type		число одиниць товару в замовленні	дане найменування товару (в кількості, визначеному атрибутом Qauntity). Замовлення може мати будь-яке число найменувань товару. Одне найменування може міститися в різних замовленнях.
Part type hierarchy	Part Type, Part Type	Б:N		Дане найменування товару є складовою частиною іншого найменування. Найменування товару може включати в себе довільне число інших найменувань. Найменування товару може бути складовою частиною довільного числа інших найменувань.
Part type attribute	Part Type, Part Type Attribute	1:Б		Товари цього найменування мають даний атрибут. Товари певного найменування можуть мати довільне число атрибутів. Конкретний атрибут може бути присутнім у одного і тільки одного найменування товарів.
Part attribute value	Part Type Attribute, Part	Б:N	Value - значення атрибута товару	Значення даного атрибута у даного товару рівне значеню атрибута Value. Для конкретного товару можуть бути задані значення довільного числа його атрибутів. Значення конкретного атрибута може бути задано для різних товарів. Для конкретного товару можна задавати значення тільки множини атрибутів, що визначається його найменуванням.

Так як СУБД, що використовується в системі VDOM (Embedded SQLite 3) не підтримує ER-модель даних, а також, в силу своїх обмежень, - і зовнішні ключі, окремим кроком проектування БД стало побудова реляційної схеми [1,2] на основі отриманої на попередньому кроці ER-схеми, згідно з правилами, описаним в [1].

Нижче наведено список правил переходу від ER-схеми до реляційної; в таблицях 2.3-2.5 представлені результати, отримані після застосування відповідних правил.

1. Кожна множина сутностей представляється самостійним відношенням, однозначні атрибути множини сутностей стають атрибутами відношення, ключі множни сутностей є можливими ключами відношень; при необхідності в якості первинного ключа відношення використовується сурогатний атрибут.

Таблиця 2.3

Результати, отримані після застосування правила 1 перетворення ER-схеми в реляційну

Вихідна множина сутностей	Атрибут множини сутностей	Вихідна таблиця	Атрибут таблиці
Partner	-	Partners	id (первинний ключ)
	Name		name
	Code		code
	isEnd-cliend		is_end_client
Location	-	Locations	id (первинний ключ)
	Name		name
	Note		note
	State		state
	City		city
	Address		address
	Zip		zip
User	-	User	id (первинний ключ)
	Name		name
	Position		position
	Phone number		phone number
	Email		email
	Password		md5_password md5 pass word (пароль зберігається в зашифрованому вигляді)
Order	-	Order	id (первинний ключ)
	Description		description
	Creation datetime		creation_datetime
	State		state
	State history		-
Part	-	Part	id (первинний ключ)
	-		-
Part Type	-	Part Type	id (первинний ключ)
	Name		Name
	Description		Description

PartTypeAttribute	-	PartTypeAttribute	id (первинний ключ)
	Name		name
	Default value		default value
	isMandatory		is_mandatory
	isKey		is_key

2. Бінарні множини зв’язків типу 1:Б без атрибутів представляються дублюванням первинного ключа 1-відношення в Б-відношення

Таблиця 2.4

Результати, отримані після застосування правила 2 перетворення ER-схеми в реляційну

Вихідна множина сутностей	Таблиця, в якій проводиться дублювання атрибута	Отриманий атрибут, дублюючий первинний ключ
Partner-User	Users	partner_id
Partner-Location	Locations	partner_id
To-location	Orders	to_location_id
From-location	Orders	from_location_id
Location-Part	Part	location_id
Part type	Part	part_type_id
Part type attribute	PartTypeAttributes	part_type_id

3. Бінарні множини зв’язків типу Б:N без атрибутів представляються самостійними відношеннями, куди дублюються первинні ключі відношень, побудованих для множин сутностей.

Таблиця 2.5

Результати, отримані після застосування правила 3 перетворення ER-схеми в реляційну

Вихідна множина зв’язків	Отримана таблиця	Отримані атрибути, дублюючі первинні ключі
Partnership	Partnerships	supplier_partner_id,
Part type hierarchy	PackageTemplates	consumer_partner_id
Part move	PartMoves	parent_part_id, child_part_id

4. Множини зв’язків з атрибутами представляються самостійними відношеннями, куди дублюються первинні ключі відношень, побудованих для

множин сутностей. Однозначні атрибути множини зв’язків стають атрибутами цього відношення.

Таблиця 2.6

Результати, отримані після застосування правила 4 перетворення ER-схеми в реляційну

Вихідна множина зв’язків	Отримана таблиця	Отримані атрибути, дублюючі первинні ключі	Отриманийі атрибут, який представляє атрибут множини зв’язків
Order line	OrderLines	order_id, part_type_id	quantity
Part attribute	PartAttributeValues	part_type_attribute_id,	value

5. Кожен багатозначний атрибут множини сутностей представляється окремим представленням, куди дублюється первинний ключ відношення, побудованого для множини сутностей; другий атрибут цього відношення призначений власне для значення.

Таблиця 2.7

Результати, отримані після застосування правила 5 перетворення ER-схеми в реляційну

Вихідний багатозначний атрибут	Отримана таблиця	Отриманий атрибут, дублюючий первинний ключ	Отриманийі атрибут, який представляє Значення атрибуту множини зв’язків
Order_history (Order)	OrderStateHistory	order_id	state, change_datetime

В результаті проведеної роботи для розроблюваної системи підтримки продажів була побудована реляційна схема бази даних, діаграму якої можна побачити на рисунку 2.5, а також в додатках.

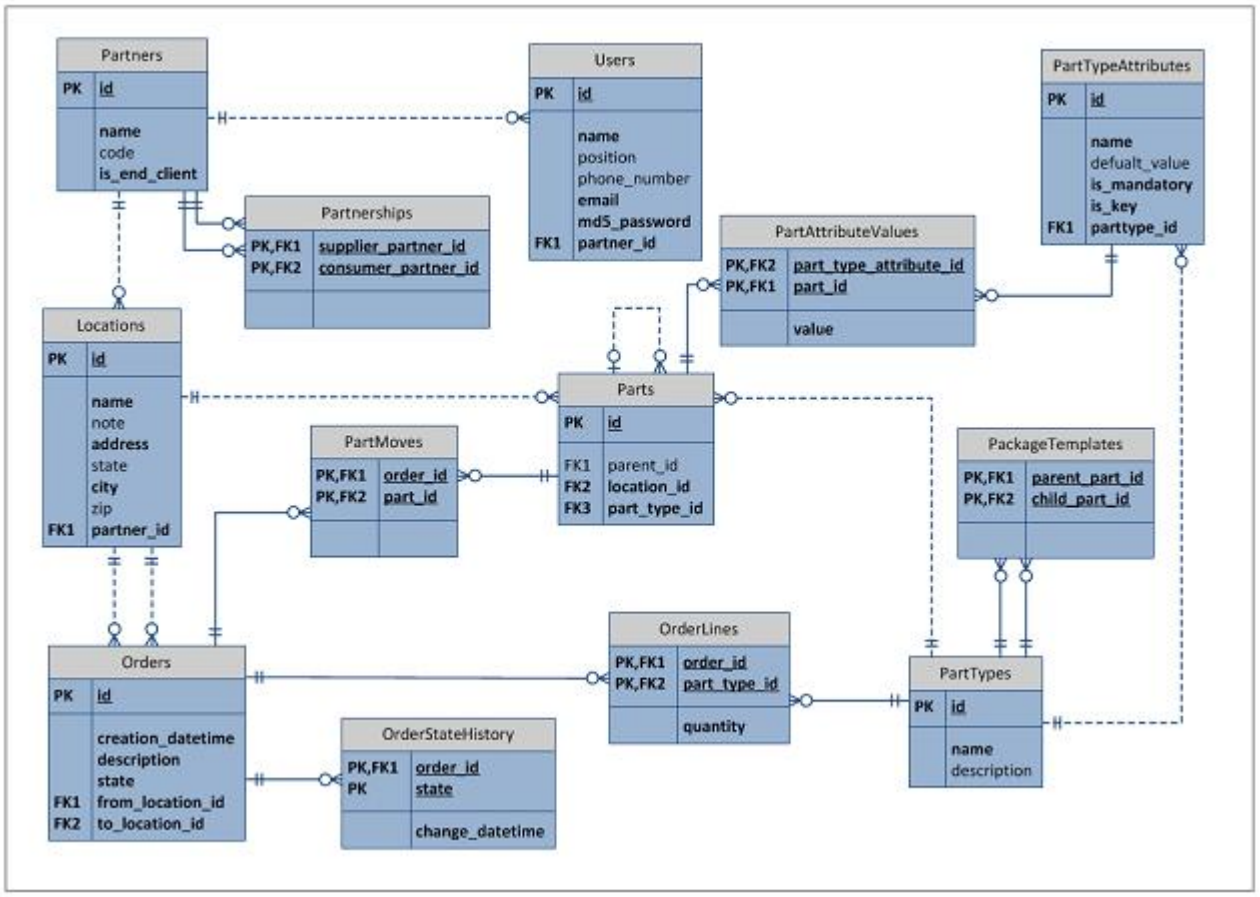


Рис. 2.5. Реляційна діаграма

Висновки до розділу 2

У цьому розділі розроблено архітектуру програмного додатку, що дозволить краще зрозуміти функції основних його частин. Створено та описано структурну схему, основними компонентами якої є: рівень клієнта, рівень бізнес-логіки та рівень даних. Описано функціональну структуру системи та її основних елементів – модулів обробки даних. Визначено основні елементи бази даних та встановлено зв’язки між ними. Спроектовано структуру бази даних.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ РЕАЛІЗАЦІЇ ТОВАРІВ НА ОСНОВІ VDOM

3.1. Технологія VDOM

Web-сервер VDOM Box є розвитком ідей технології VDOM (Visual Dynamic Object Model, візуальна динамічна об’єктна модель) [6] – клієнт серверної системи, призначеної для створення і управління сайтом за допомогою спеціалізованих візуальних засобів. Технологія являє сайт у вигляді об’єктів призначених для користувача інтерфейсу і, по суті, реалізує компонентну модель розробки web-додатків. Технологія VDOM розроблена CyberTronique inc. з метою допомогти організаціям і користувачам розробляти web-додатки без глибоких знань в web-технологіях.

Для створення сайту в системі VDOM користувач використовує спеціальний web-додаток, яке включає в себе такі інструменти, як WYSIWYG-редактор для сторінок (рисунок 3.1), спеціальні засоби для проектування схеми програми та інтерфейс для розробки сценаріїв, що вбудовуються в сайт.

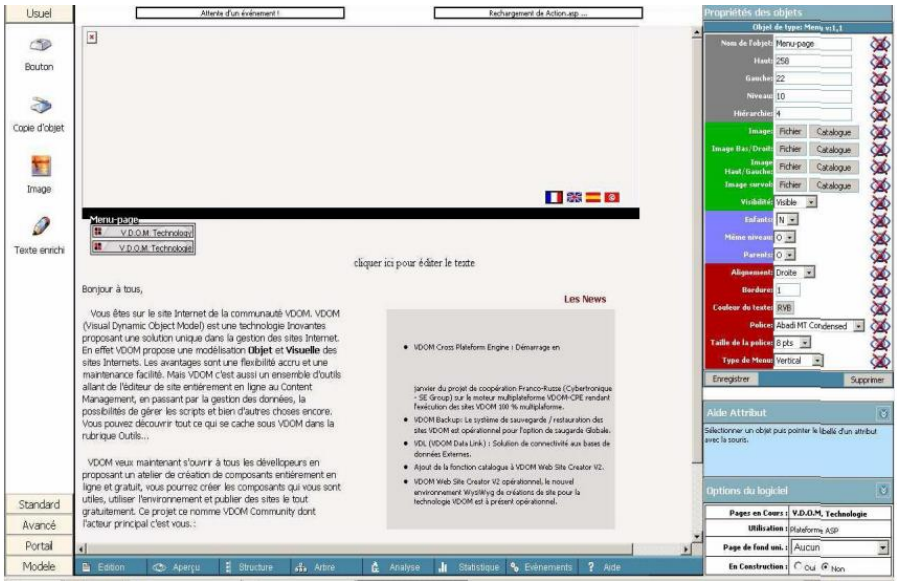


Рис. 3.1. Інтерфейс візуального редактора VDOM

Система E2VDOM [7] є другим поколінням системи VDOM. У ній була представлена можливість управляти не тільки візуальною складовою сторінок, але і також логікою їх роботи. Логіка реалізується за рахунок використання системи event-> Action. Кожен об’єкт крім атрибутів має певний набір подій, на які він може реагувати. Наприклад, для кнопки (тип Button) такою подією може бути OnClick, генерує, коли користувач клацає по кнопці. Конкретна подія може бути пов’язана з певним оброблювачем (action-м).

Розрізняють клієнтські і серверні action-и. Клієнтські action-и виконуються на стороні клієнта і не потребують будь-якого втручання з боку сервера. Особливість серверних action-ів полягає в тому, що вони виконуються на VDOM сервері. Спочатку сигнал про те, що на стороні клієнта спрацювала якась подія (наприклад, користувач натиснув на кнопку) відправляється на сервер, де виконується його обробка. Якщо в ході обробки події (виконання скрипта на мові Python), у будь-яких об’єктів змінилися значення атрибутів, сервер відправляє клієнту HTML код, який відповідає цим зміненим об’єктам. Тобто, по-перше, обробку всієї логіки здійснює сервер, що знижує навантаження на стороні клієнта. По-друге, при зміні значень атрибутів окремих об’єктів (тобто їх візуальній складовій) виконується повторне відображення тільки цих об’єктів, а не всієї сторінки цілком.

Детальна схема роботи системи event->action приведена на рисунку 3.2.

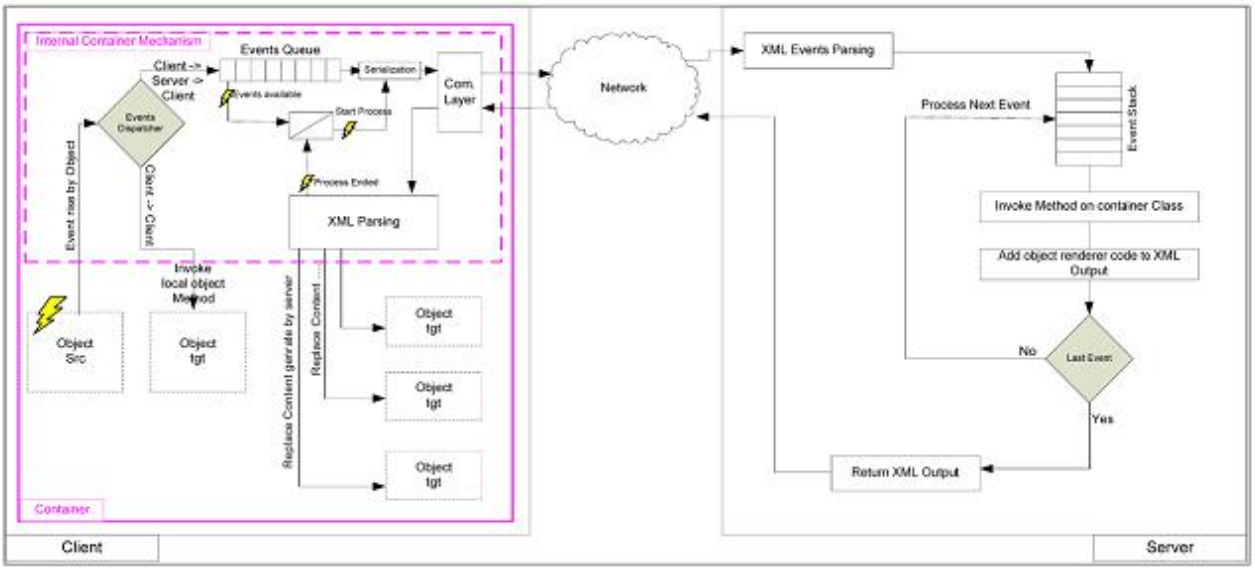


Рис. 3.2. Схема роботи системи event->action платформи VDOM

3.2. Програмна реалізація системи реалізації товарів

У попередньому розділі була розглянута загальна структура класів шару бізнес-логіки, а також відповідна UML діаграма класів. Завдання реалізації даного шару полягає в написанні класів на мові Python [8] відповідно до отриманої діаграми. В даному пункті розглянуті особливості реалізації загальної структури класів, а також рішення таких задач, як контроль обмежень цілісності, перетворення типів даних між БД і шаром бізнес-логіки, обмеження області видимості «закритих getter-ів» класу (описаних в попередньому розділі). Значну частину типового класу шару бізнес-логіки становлять методи, які виконують ORM-перетворення. Структура подібних методів передбачає певну обробку даних відповідно до їх логіки і, природно, звернення до бази даних. Взаємодія з БД відбувається за рахунок побудови запитів на мові SQL і передачі їх СУБД через стандартний інтерфейс. Розглянемо інтерфейс платформи VDOM для роботи з базою даних. Він включає в себе три основні функції:

```
vdom_sql_fetch_one(dbase_guid, sql_query, params)
vdom_sql_fetch_all(dbase_guid, sql_query, params)
vdom_sql_commit(dbase_guid, sql_query, params)
```

Перші дві функції призначені для вибірки даних з БД і приймають параметри select-запиту. Відрізняються ж вони тим, що vdom_sql_fetch_one повертає лише перший рядок результату переданого SQL-запиту у вигляді Python-списку, тоді як vdom_sql_fetch_all повертає всі рядки запиту у вигляді списку списків. Третя функція, vdom_sql_commit, призначена для внесення змін в БД і приймає insert-, update- і delete-запити. Для insert- і update-запитів повертається значення первинного ключа останнього вставленого/оновленого записи. Для delete-запиту повертається None.

Зауважимо, що згадані функції мають аналогічний набір параметрів. Розглянемо їх детальніше:

- dbase_guid - GUID локальної бази даних програми.

- sql_query - рядок запиту на мові SQL.

- params - кортеж значень параметрів зазначеного SQL-запиту.

Рядок SQL-запиту може містити довільне число параметрів зазначених знаком «?», список значень цих параметрів вказується в кортежі params. Даний механізм роботи з параметрами покликаний виключити можливість такої відомої уразливості додатків як SQL-ін’єкція. Наведемо приклад звернення до бази даних через стандартний інтерфейс.

```
db_rows = vdom_sql_fetch_one("426f6f74-0000-11aa-aa11-00306543ecac",
                             "SELECT * FROM users WHERE id=?", (45, ))
```

Даний метод поверне Python-список значень атрибутів записів таблиці users зі значенням id, рівним 45. Зауважимо, що база даних в типовому VDOM-додатку, як правило, одна, а значить, кожен раз вказувати її GUID при виклику стандартних функцій не дуже зручно.

Дану незручність можна подолати за рахунок використання класу (назвемо його Database), який би зберігав посилання на локальну БД. Інтерфейс даного класу практично повністю повторює стандартний VDOM-інтерфейс, але усуває необхідність щоразу вказувати GUID бази даних. Взагалі кажучи, введення подібного класу є стандартним прийомом, використовуваним розробниками на VDOM-платформі. Наведемо приклад звернення до бази даних через проміжний інтерфейс.

```
db_rows = Database.maindb().fetch_all("SELECT id, name FROM partners")
```

Розглянемо тепер рішення такого важливого завдання, як контроль обмежень цілісності. Контроль відбувається всередині методу save() кожного класу. Спочатку виконується перевірка всіх умов цілісності які можуть бути порушені при збереженні поточного об’єкта в БД, після чого виконується вже безпосередньо збереження об’єкта.

При порушенні будь-якого обмеження цілісності генерується виняток типу VBILogicError з відповідним описом помилки. Даний клас (успадкований

від класу Exception) був введений для розмежування помилок, породжуваних порушенням предметної області, і помилок іншого плану.

Передбачається, що при необхідності збереження об’єкта клієнт шару бізнес-логіки буде поміщати виклик методу save() всередину блоку try ... except і належним чином обробляти виняткову ситуацію типу VBILogicException, наприклад, виводячи отриманий опис помилки користувачеві у спливаючому вікні.

Зауважимо, що хоча СУБД Embedded SQLite 3 і підтримує обмеження цілісності типу UNIQUE і NOT NULL, на жаль, при зверненні до бази даних через стандартний інтерфейс, клієнту не видається інформація про те, який саме тип був порушений. Так як клієнтові шару бізнес-логіки така інформація може бути цікава, перевірка згаданих обмежень цілісності також були реалізовані на рівні шару.

Нагадаємо, що СУБД Embedded SQLite має обмежений набір типів даних, тому деякі стовпці таблиць містять дані в «неправильному» форматі. Наприклад, стовпець з даними «дата / час» зберігає їх у вигляді рядка, дані типу boolean зберігаються як число і так далі. У класах бізнес-логіки хотілося б бачити в якості значень атрибутів дані «правильних» типів. Таким чином, постає завдання перетворення типів даних між БД і шаром бізнес-логіки. Тепер розглянемо, в яких методах необхідно виконувати дані перетворення. Зміна об’єктів бізнес-логіки відбувається всередині методів insert() і update(), а їх створення по рядку вибірки з БД - всередині методу fill_from_row (row). Таким чином, виконання перетворень потрібно активувати саме в ці три методи, що і було зроблено. Наприклад, метод fill_from_row (row) класу Partner виглядає наступним чином:

```
@classmethod
def fill_from_row(self, row):
        partner = Partner()
        partner.id = row[0]
        partner.name = row[1]
        partner.is_end_client = bool(row[2])
        return partner
```

Тепер поговоримо про реалізацію області видимості «закритих getter-ів» класу (до них відносяться методи вибірки об’єктів даного класу, призначені для використання в методах інших класів бізнес-логіки). Тут необхідно було вирішити задачу обмеження області видимості даних методів, зробивши їх доступними для класів шару бізнес-логіки й недоступними за його межами. Мова Python не має засобів, аналогічним, наприклад, механізму дружніх класів в мові C ++. З огляду на ці обставини, довелося зробити розглянуті методи відкритими, а також ввести умовну угоду про їх невикористання за межами шару бізнес-логіки. Розглянемо задачу «розміщення» класів шару бізнес-логіки. На платформі VDOM існує поняття «бібліотеки» (library). Окремо взята бібліотека являє собою звичайний Python-модуль, тобто файл вихідного коду на Python, який може бути використаний як в інших VDOM-бібліотеках, так і в серверних скриптах (є, як правило, обробники певних VDOM-подій на сторінці) за допомогою стандартного механізму імпорту мови Python. Таким чином, механізм бібліотек на платформі VDOM реалізує можливість повторного використання коду.

Реалізація кожного класу шару бізнес-логіки була оформлена у вигляді окремої бібліотеки. Була використана наступна схема іменування: class_ <назва класу>. Наприклад бібліотека, яка реалізує клас Partner, називається class_partner. Безпосередньо для написання класів використовувався редактор скриптів VDOM Script Editor, що є певним доповненням до основного середовища розробки - VDOM IDE, яке, звичайно ж, теж підтримує можливість редагування Python-коду, але, на жаль, не кращим чином - в ній відсутнє навіть підсвічування синтаксису, не кажучи вже про інші «зручності» для розробника.

Що стосується тестування працездатності розроблених класів, то в умовах відсутності на даному етапі для користувача web-інтерфейсу воно виконувалося на прості тестовій сторінці (містить кілька кнопок і візуальних компонентів для введення/виведення даних). В результаті проведеної роботи були реалізовані 6 класів бізнес-логіки: Partner, User, Location, Order, Part і PartType. Фрагмент коду реалізації одного з класів представлено на рисунку 3.3

```
90      @classmethod
91      def get_from_by_order(self, order):
92          """ gets source location (from_location) of the specified order """
93          db_row = Database.maindb().fetch_one(
94              """
95              SELECT
96                  location.id,
97                  location.street,
98                  location.state,
99                  location.city,
100                 location.note,
101                 location.name,
102                 location.zip,
103                 location.partner_id
104             FROM `order`
105             JOIN location ON `order`.from_location_id = location.id
106             WHERE `order`.id = ?
107             """, (order.id, ))
108         return Location().__fill_from_row(db_row)
109
110     @classmethod
111     def get_by_id(self, id):
112         db_row = Database.maindb().fetch_one(
113             """
114             SELECT id, street, state, city, note, name, zip, partner_id
115             FROM location
116             WHERE id = ?
117             """, (id, ))
118         return Location().__fill_from_row(db_row) if db_row else None
119
120     def get_partner(self):
121         """ gets a partner who owns the current location """
122         from class_partner_new import Partner
123         return Partner.get_by_location(self)
```

Рис. 3.3. Фрагмент реалізації класу Location шару бізнес-логіки

Специфікація зовнішнього API для розроблюваної системи VBI Partners була коротко описана в попередньому розділі. В даному пункті розглянута реалізація API на платформі VDOM. Розглянемо для початку засоби, що надаються використовуваною платформою для вирішення поставленого завдання. Платформа VDOM надає можливість віддаленого виклику процедур за допомогою бібліотеки, яка працює по протоколу SOAP. Кожна процедура оформляється у вигляді окремого файлу вихідного коду на мові Python і поміщається в додаток, з якого може бути викликана. Введення і виведення даних здійснюється у форматі XML. Процедура отримує дані через сесію, через змінну з фіксованим ім’ям (xml_data). Вихідні дані відправляються аналогічно - зберігаються в змінну з фіксованим ім’ям (Response).

Тепер поговоримо безпосередньо про реалізацію зовнішнього API. Розробка кожної окремої API-операції передбачає послідовне рішення трьох завдань, а саме:

- Аналіз вхідних XML-даних.
- Реалізація логіки операції.
- Формування вихідних XML-даних.

Логіка операцій API зводилася до використання шару бізнес-логіки та деякій додатковій обробці, специфічною для кожної окремої операції. Завдання формування вихідних XML-даних вирішувалося за допомогою такої структури Python, як шаблони рядків (template strings). Наприклад:

```
template_response = "<success>\n\t<user_id>%s</user_id>\n<success>"
request.session.value("response", template_response % user.id)
```

Задача аналізу XML, вирішувалася за допомогою стандартної бібліотеки Python - minidom, що надає всі необхідні для вирішення даного завдання засоби. Розглянемо згадану бібліотеку більш детально. Аналіз XML-даних починається з виклику функції parseString, яка повертає екземпляр класу Document, що представляє якусь внутрішню модель XML-документа. Відповідно до даної моделі документ має ієрархічну структуру вкладених вузлів (примірників класу Node). Існують два основних види вузлів: вузли-елементи (elements) і текстові вузли (Text nodes). Вузли-елементи представляють тег XML-документа і відповідно можуть містити довільне число дочірніх вузлів різного виду. Текстові вузли представляють текст, укладений всередині тегів XML-документа. Зауважимо, що будь-яка послідовність символів, укладених між межами тегів (в тому числі містить прогалини, символи переходу на новий рядок і інші), утворює текстовий вузол. У наступному прикладі тег <info> містить три вузли: текстовий ("\ N \ t"), елемент <name>, і ще один текстовий ("\ n").

```
<info>
    <name>John</name>
</info>
```

Отримати кореневий елемент XML-документа можна, викликавши метод getDocumentElement() екземпляра класу Document.

Так як об’єкти класу Node є ключовими в структурі XML-документа, розглянемо коротко його інтерфейс. Об’єкт класу Node має наступні атрибути:

- nodeType - тип вузла (має значення ELEMENT_NODE для вузла-елемента і TEXT_NODE для текстового вузла).

- tagName - назва тега, яку представляють елементом (для текстового вузла завжди None).

- attributes - словник атрибутів відповідного тега, ключами якого є назви атрибутів, а значеннями - значення цих атрибутів (для текстового вузла завжди None).

- childNodes - список дочірніх вузлів поточного елемента (для текстового вузла завжди порожній список).

- nodeValue - строкове значення текстового вузла (для елемента завжди None).

Також об’єкт класу Node має метод getElementsByTagName (tagName), що видає список його нащадків (як прямих, так і непрямих) з вказаною назвою тега. Можна умовно виділити дві стратегії аналізу XML-документа. Перша з них передбачає рекурсивний обхід XML-дерева, починаючи від кореневого елемента, і отримання для текстових вузлів їх строкових значень, а для елементів значень їх атрибутів і списку їх нащадків. Якщо ж в документі свідомо відсутні теги з однаковою назвою, то завдання значно спрощується. Можна просто викликати метод getElementsByTagName від кореневого елемента і аналізувати отримані елементи - в цьому полягає ідея другого підходу.

В силу особливостей специфікації розробляється API (відсутність повторюваних тегів в структурі вхідних даних кожної операції) при аналізі XML-даних був використаний другий підхід (тобто передбачає використання методу getElementsByTagName). Розглянемо тепер питання «розміщення» операцій зовнішнього API. Для забезпечення можливості віддаленого виклику

операції вона повинна бути реалізована в вигляді окремого Python-скрипта і поміщена в певний HTML-контейнер. Відповідно до цієї умови в додатку був створений HTML-контейнер з ім'ям «API», всередину якого були поміщені скрипти, що реалізують відповідні операції. Іменування скриптів вироблялося відповідно до назв реалізованих операцій.

Для тестування API було написано невеликий VDOM-додаток, який дозволив встановлювати зв'язок з конкретним сервером, запускати з нього віддалені процедури і відстежувати повернені ним результат. При розробці додатку використовувався стандартний Python-клас VDOMService, що представляє собою певну «обгортку» над згаданою SOAP-бібліотекою.

Розглянемо інтерфейс даного класу. Він включає в себе два методи, а саме:

```
@staticmethod
connect(url, login, md5_hexpass, application_id)
```

Даний метод встановлює з'єднання з зазначеним VDOM сервером, а точніше з конкретним додатком, встановленим на сервері, і повертає екземпляр класу VDOMService, готовий до роботи. Метод має наступні параметри:

- url - IP адреса VDOM сервера.

- login - логін для входу на сервер.

- md5_hexpass - md5-хеш пароля для входу на сервер.

- application_id - GUID додатку, з яким виконується з'єднання.

```
call(self, container_id, action_name, xml_data)
```

Даний метод здійснює виклик віддаленої процедури і повертає XML дані, які є результатом її виведення. Розглянемо його параметри:

- container_id - GUID контейнера, що містить вилучену процедуру.

- action_name - ім'я віддаленої процедури.

- xml_data - XML дані, що подаються на вхід віддаленої процедури.

Відповідно до специфікації замовника в результаті проведеної роботи було реалізовано і протестовано 7 віддалених процедур, а саме:

- login - авторизувати користувача в системі.
- create_end_client - створити клієнта зазначеного партнера.
- create_user - створити користувача вказаного партнеру.
- create_location - створити склад вказаного партнера.
- create_order - створити замовлення.
- get_locations - отримати список складів, що належать зазначеному партнеру.
- get_part_types - отримати список всіх найменувань товару.

На рисунку 3.4 наведено фрагмент вихідного коду однієї з операцій.

```
22      # password
23      password_tags = params.getElementsByTagName("password")
24      if not(password_tags and password_tags[0].firstChild):
25          raise Exception("no info on password")
26
27      # partner id
28      partner_guid_tags = params.getElementsByTagName("partner_guid")
29      if not(partner_guid_tags and partner_guid_tags[0].firstChild):
30          raise Exception("no info on partner guid")
31
32
33      #       create user     #
34
35      username = username_tags[0].firstChild.nodeValue
36      password = password_tags[0].firstChild.nodeValue
37      partner = Partner.get_by_guid(partner_guid_tags[0].firstChild.nodeValue)
38      if not partner:
39          raise Exception("partner with such guid does not exist")
40
41      # final
42      new_user = partner.create_user(username)
43      new_user.password = password
44      new_user.save()
45      request.session.value("response", success_template % new_user.id)
46
47  except ExpatError:
48      request.session.value("response", api_error(PARSING_FAILED))
49  except Exception as e:
50      request.session.value("response", error_template % str(e))
```

Рис. 3.4. Фрагмент вихідного коду операції create_user

Для кожної окремої сторінки необхідно було розробити як візуальну, так і поведінкову частини. Розглянемо рішення двох даних завдань. Візуальна частина розроблялася за рахунок розміщення на кожній сторінці об’єктів та значень їх атрибутів. У деяких місцях для задання оформлення, використовувалися стилі CSS[10]. Різні, чисто декоративні елементи оформлення, такі, як наприклад, логотип, іконки, рисунки, шапка сайту, використані при оформленні користувальницького інтерфейсу, були розроблені

дизайнером. Для завдання візуальній складовій сторінок, використовувалася інтегроване середовище розробки VDOM IDE.

Поведінкова частина розроблялася за рахунок використання технології event-> Action. Для побудови схеми взаємодії розміщених на сторінках об’єктів також використовувалася VDOM IDE. Для написання логіки обробників подій (action-ів) використовувалася VDOM Script IDE, так як функціональність редактора скриптів VDOM IDE виявилася досить-таки обмеженою.

Всього в результаті проведеної роботи відповідно до специфікації замовника було розроблено 17 web-сторінок:

- Login.
- Dashboard.
- Location List.
- Location Details.
- Edit Location.
- Order List.
- Order Details.
- Edit Order.
- Assign Order.
- Partner List.
- Partner Details.
- Part List.
- Part Details.
- Search Parts.
- Add Part.
- Assemble Package.
- System List.

Опис і зовнішній вигляд даних сторінок представлено в наступному розділі роботи. Розглянемо приклад розробки одній зі сторінок web-інтерфейсу.

Розглянемо процес розробки web-інтерфейсу на прикладі сторінки Assign Order, зовнішній вигляд якої можна побачити на рисунку 3.3. Дана сторінка

дозволяє користувачу здійснювати резервування товарів відповідно до конкретного замовлення. Розглянемо процес розробки візуальної і поведінкової частин web-сторінки.

Для завдання візуальній складовій сторінки були використані наступні типи об’єктів:

- RichText. Використовується для відображення текстових даних. Здатний розпізнавати HTML теги.

- Button. Являє собою такий стандартний елемент керування як кнопка.

- Container. Призначений для логічного і візуального об’єднання декількох об’єктів.

- Copy. Дозволяє «копіювати» певний об’єкт, розташований на іншій сторінці. Використовується у випадках, коли необхідно візуально продублювати один об’єкт на багатьох сторінках (наприклад, заголовок, загальний для декількох сторінок).

- Datatable. Використовується для виведення даних в табличній формі. Здатний відслідковувати натискання користувача на рядки таблиці.

- Growl. Призначений для видачі повідомлень у вигляді спливаючого вікна.

- Об’єкт Copy використовується для розміщення заголовка вгорі сторінки (заголовок заздалегідь скомпонований і зберігається на окремій сторінці Design). Об’єкти типу RichText використовуються для нанесення декоративних і пояснювальних написів. Верхня частина з інформацією реалізована за допомогою об’єкта Container, що містить деякі інші об’єкти. Кнопки в самому низу сторінки є екземплярами типу Button. Два нижніх списки товарів представлені об’єктом DataTable. Варто відзначити, що тип DataTable є ключовим для всього призначеного для користувача інтерфейсу розроблюваної системи VBI Partners, так як більшість даних відображається саме в табличній формі.

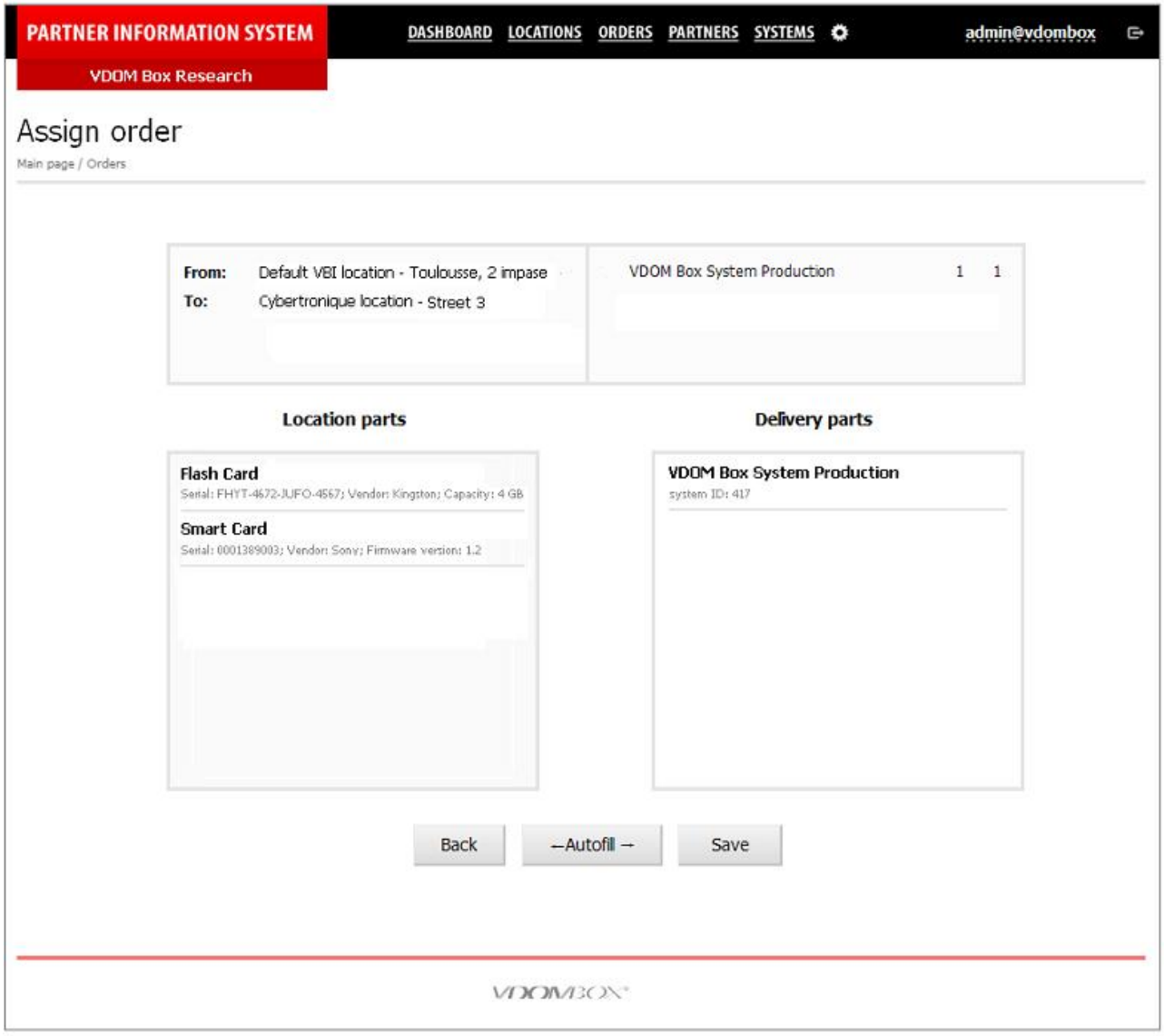


Рис. 3.5. Загальний вигляд сторінки Assign Order

Розглянемо, як задається зовнішній вигляд об’єкта визначенням значень його атрибутів, на прикладі одного зі списку товарів (зауважимо, що обидва списки мають однаковий зовнішній вигляд, а відповідно і набір значень атрибутів). Значення атрибутів відповідного об’єкта типу DataTable наведені в таблиці 3.1

Таблиця 3.1

Список атрибутів об’єкта loc_parts_table і їх значень

Назва	Значення	Опис атрибутів
Name	loc_parts_table	Використовується для ідентифікації об’єкта.
Hierarchy	0	Визначає порядок відображення об’єкта на сторінці.
Visibility	Yes	Визначає видимість об’єкта.
Z-index	0	Глибина (Z-координата) об’єкта на сторінці.
Left	180	Положення об’єкта по осі X від лівого краю його контейнера
Top	355	Положення об’єкта по осі Y від верхнього краю його контейнера
Width	250	Ширина об’єкта.
Height	300	Висота об’єкта.
Title		Тема, що відображається зверху таблиці.
Header	["id", "info"]	Шапка таблиці. Визначає кількість стовпців таблиці і їх імена.
Show Header	No	Визначає видимість шапки таблиці.
Data	Задається динамічно	Дані, які відображаються в таблиці.
Key	“id”	Ім’я поля, яке є ключовим для таблиці
Hidden Fields	["id"]	Список прихованих полів
Selection Mode	Single-Select	Режим вибору рядків таблиці. Single-Select: обраною може бути тільки одна строка Multi-Select: обраними можуть бути кілька рядків.
Style		Дозволяє задати оформлення таблиці через CSS-стилі.
Skin	“Ligh Silver”	Скін, що визначає зовнішній вигляд таблиці.

Логіка додавання (видалення) товарів в замовлення, а також збереження зроблених змін задається шляхом призначення обробників (скриптів на мові Python) певним подіям. У таблиці 3.2 наведено схему реалізації логіки роботи розглянутої сторінки. На рисунку 3.6 можна побачити, як ця логіка реалізована в інтегрованому середовищі розробки VDOM IDE.

Таблиця 3.2

Схема логіки роботи сторінки Assign Order

Назва об’єкта	Назва події	Опис події	Назва обробника
loc_parts_table	rowclick	Користувач виділив рядок таблиці	add_part_to_delivery
order_parts_table	rowclick	Користувач виділив рядок таблиці	remove_part_from_delivery
save_button	click	Користувач натиснув кнопку	save_delivery

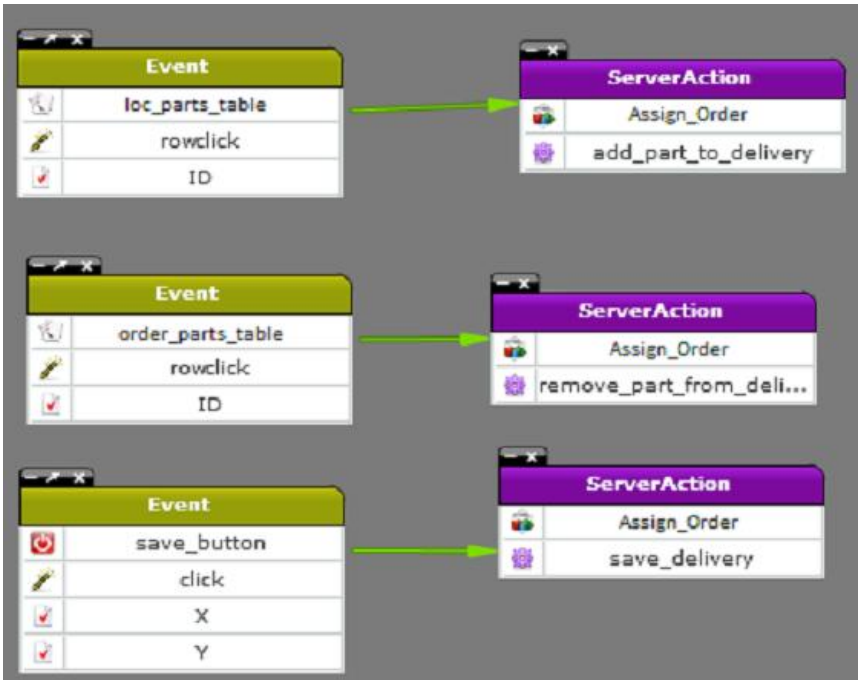


Рис. 3.6. Схема логіки роботи сторінки Assign Order, яка відображається в середовищі VDOM IDE

Web-сервіси. Щоб маніпулювати даними на сервері, створені web-сервіси, які дозволяють виробляти базові операції над об’єктами в пам’яті. На базі цього механізму можна створювати сторонні додатки, які будуть виробляти адміністрування системи або розширювати її функціональність. Наприклад, IDE-редактор, що розробляється як основний засіб розробки дизайну для додатків на VDOM Box Server, заснований на цих web-сервісах.

Web-сервіси - це програмні модулі, що працюють по протоколу SOAP, які надають віддаленим програмами функціональність, описану на мові XML. Взаємодія веб-сервісів і сторонніх продуктів відбувається за протоколом SOAP (надбудова над протоколом HTTP). Для полегшення створення додатків для VDOM Box реалізована бібліотека, яка прозоро для програміста представляє роботу з об’єктами на сервері.

Розглянемо основні принципи взаємодії з сервером. Стандартний механізм веб-сервісів не має механізму збереження даних про користувача між окремими транзакціями. З цієї причини, щоб забезпечити безпеку і зручність спілкування клієнта з сервером, реалізований механізм «сесій» на зразок механізму, реалізованого в багатьох браузерах. При відкритті з’єднання за допомогою сервісу open_session поточному клієнту присвоюється номер сесії, і подальше спілкування відбувається з його використанням. Закінчення сесії проводиться за допомогою сервісу close_session. Для запобігання перехоплення сесій передбачений механізм динамічного ключа - рядки, які ідентифікують поточну транзакцію в діалозі. Даний ключ обчислюється на підставі призначеного для користувача паролю або номеру повідомлення і забезпечує базовий захист від перехоплення сесії.

3.3. Програмна реалізація бази даних

У попередньому розділі роботи була описана реляційна схема бази даних системи VBI Partners. В даному підрозділі мова піде про розробку самої бази даних на платформі VDOM на основі отриманої схеми.

Для початку розглянемо технічні можливості VDOM по розробці БД. Як вже було сказано раніше, в системі VDOM в якості СУБД використовується варіація SQLite - Embedded SQLite. Дана СУБД «вбудовується», тобто розроблена з її допомогою база даних інтегрується безпосередньо в сам додаток.

Такий підхід до організації БД хороший на етапі розробки. Він дозволяє, не відволікаючись на технічні деталі, скоротити час розробки бази даних і відповідно самого додатку. Крім того, при такому підході не потрібно розгортати окремий сервер з СУБД і налагоджувати зв'язок додатку з базою даних. Досить просто встановити додаток на VDOM сервер, і він буде повністю готовий до роботи.

Однак за дані зручності доводиться розплачуватися відносно низькою швидкодією бази даних, а також урізаною функціональністю самої СУБД. До найбільш значущим її обмежень відносяться:

- Відсутність підтримки складових первинних ключів.
- Відсутність підтримки зовнішніх ключів.
- Відсутність підтримки каскадних операцій.
- Відсутність підтримки тригерів.
- Обмежений набір типів даних.

Розглянемо, як проходить розробка бази даних на платформі VDOM. Для створення і редагування бази даних використовується інтегроване середовище розробки VDOM IDE, що є основним засобом розробника на VDOM-платформі.

Оскільки база даних вбудовується безпосередньо в додаток, то для початку необхідно створити як мінімум порожній додаток. Далі в ньому створюється так званий Database container, що представляє локальну базу даних поточного додатку. Створення таблиць здійснюється шляхом додавання в контейнер примірників VDOM-типу Database table. Редагування структури таблиці (тобто її стовпців) здійснюється шляхом редагування атрибуту schema за допомогою спеціальної форми.

Для кожного стовпця можна задавати назву, тип даних, а також обмеження цілісності типу UNIQUE і NOT NULL. Зауважимо, що для кожної створеної таблиці автоматично генерується сурогатний автоінкрементний стовпець з ім'ям «id», який є первинним ключем таблиці. Схема редагованої бази даних відображається в головному вікні середовища розробки. Зміна

даних, що зберігаються в таблиці, здійснюється шляхом редагування атрибута data через відповідну форму.

В результаті проведеної роботи на основі побудованої реляційної схеми булла розроблена база даних. Відповідну реляційну діаграму представлено на рисунку 3.7. Перше, що кидається в очі, відсутність на діаграмі зв’язків між таблицями (не підтримуються зовнішні ключі).

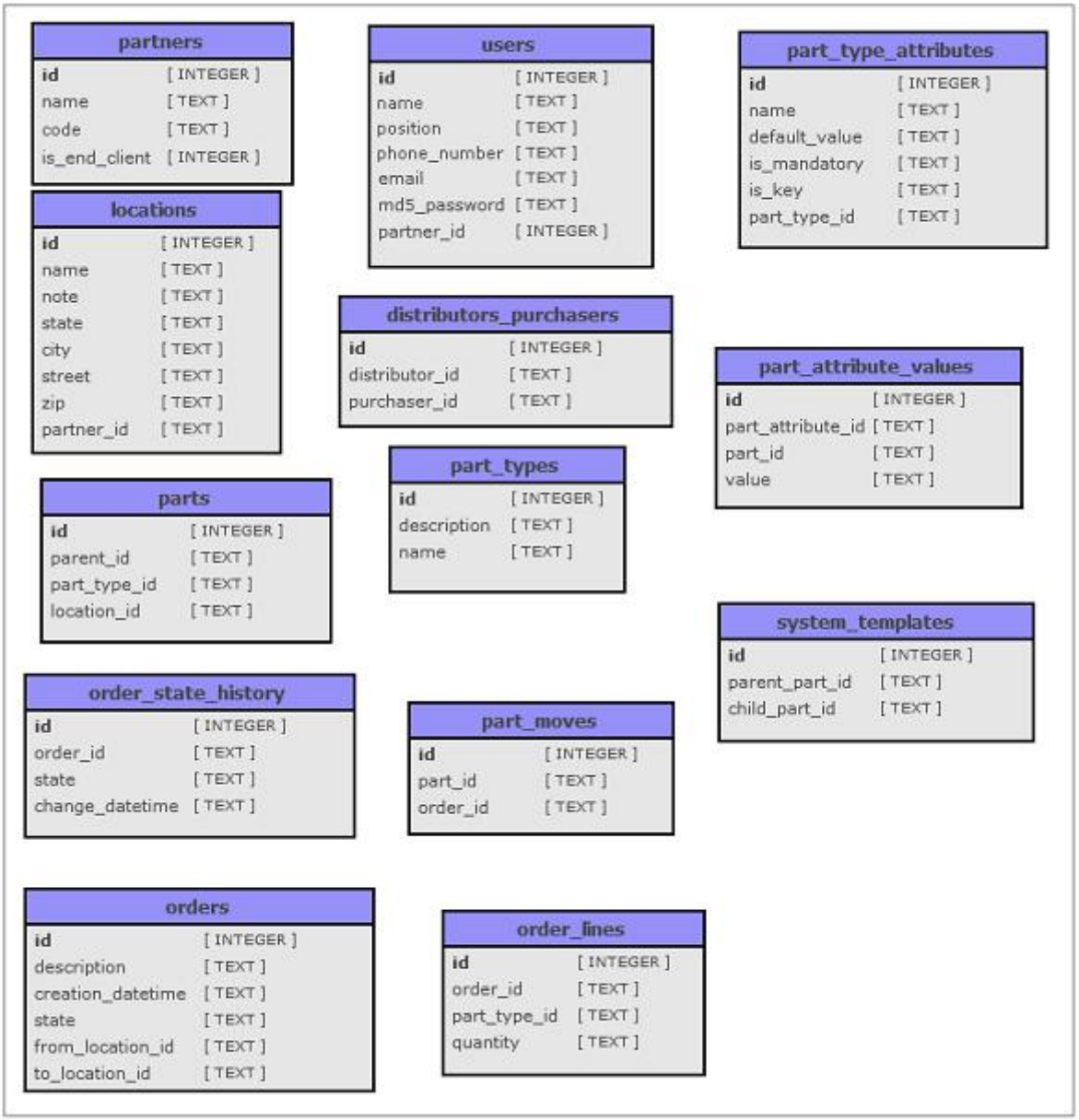


Рис. 3.7. Реляційна діаграма, що відображається середовищем VDOM IDE

В цілому, незважаючи на описані обмеження використовуваної СУБД, база даних виявилася цілком придатною для системи VBI Partners. Проблему малого числа типів даних вдалося вирішити зведенням необхідних типів до наявних.

Наприклад, можна звести Boolean до Integer (0 - false, 1 - true), datetime - до string і так далі. Завдання контролю обмежень цілісності, специфічних для предметної області (відсутність підтримки тригерів), а також обмежень цілісності посилань (Відсутність підтримки зовнішніх ключів) вдалося повністю вирішити на рівні шару бізнес-логіки. Що ж стосується проблеми низької швидкодії, то, за словами замовника, вона не є критичною, принаймні, на початковому етапі експлуатації системи.

По завершенню розробки самої бази даних, в неї були внесені деякі початкові дані, надані замовником. Ці дані містили інформацію про основних «дійових осіб» (партнери, їх склади, користувачі), а також про надані замовником товари і послуги.

Висновки до розділу 3

У даному озділі обґрунтовано технологію, мову програмування та розроблено програмну систему. Обґрунтовано засоби розробки бази даних та створено програмну реалізацію бази даних програмної системи за допомогою механізму збережених процедур.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ДОСЛІДНА ЕКСПЛУАТАЦІЯ

4.1. Тестування

Інфраструктура модульного тестування, така як JUnit, дозволяють тестувати код, що виконується на стороні сервера. Однак в типовому Web-додатку серверний код є лише малою частиною всього коду програми. У таких програмах може бути багато коду, який можна протестувати тільки за допомогою інструменту, який працює з браузером.

Одним з найбільш складних аспектів тестування web-додатків є тестування частини коду програми, що відноситься до призначеного для користувача інтерфейсу, яка, як правило, генерується з HTML і JavaScript-коду. Цей код виконується в браузері, а не в процесі сервера, тому його можна перевірити тільки за допомогою браузера. Прикладами такого коду є сторінки JavaServer Pages (JSP), код PHP і Ruby.

У роботі для тестування розробленого додатку використано інструмент Selenium, за допомогою якого можна створювати і автоматизувати тести web-додатків.

Selenium RC є частиною набору інструментів, наявних в проекті Selenium. З його допомогою можна автоматично запускати створені тести. Selenium RC працює на багатьох операційних системах і з різними браузерами, в тому числі з Windows® Internet Explorer®, Mozilla Firefox і Opera.

Використовуючи Selenium RC, можна автоматично виконувати тести скільки завгодно раз. Крім того, цей інструмент дозволяє створювати більш складні тести, ніж ті, які можна робити в Selenium IDE. В тести можна додавати інструкції умов і повторів, які можуть стати в нагоді, якщо ви хочете виконати тести з певним набором даних. Також можна обробляти очікувані виключення з

допомогою конструкцій, наявних в інфраструктурах модульного тестування, таких як JUnit.

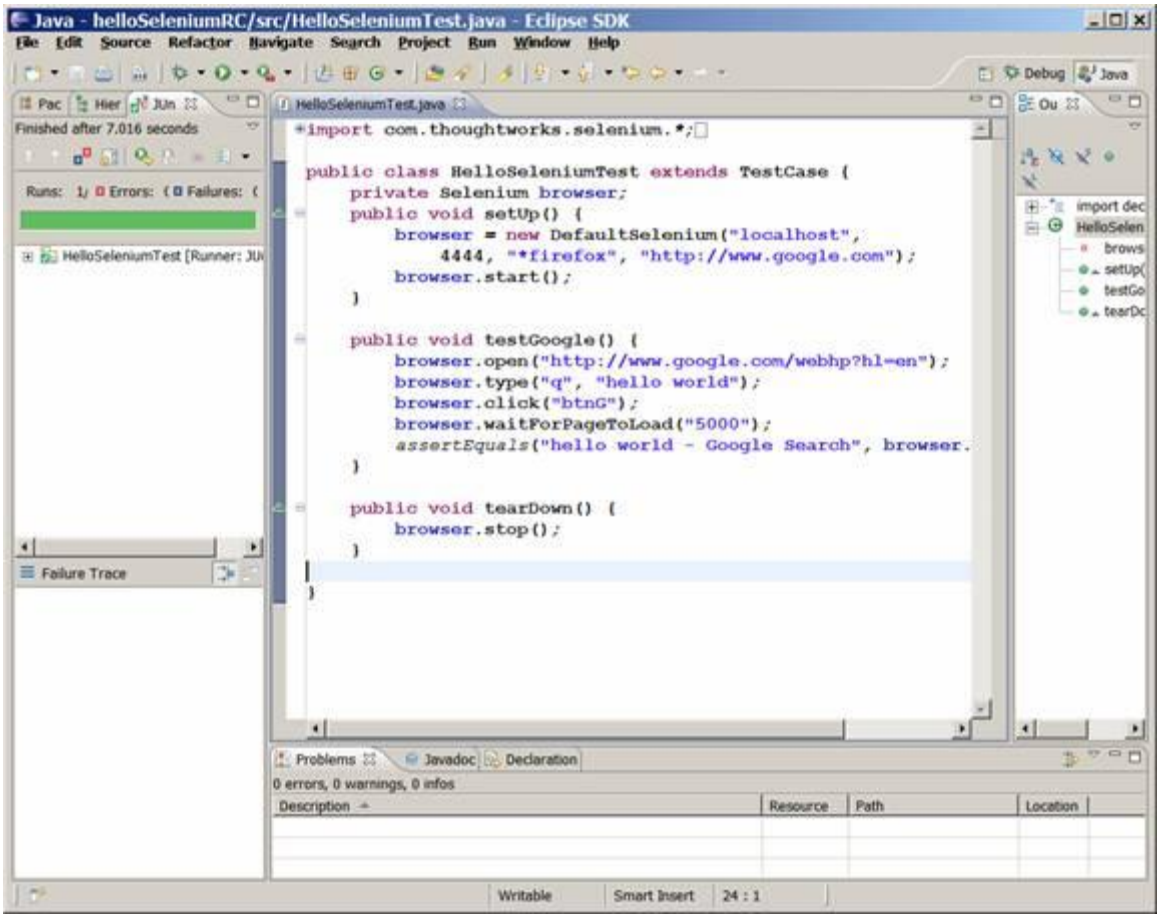


Рис. 4.1. Тестування веб-додатків за допомогою Selenium RC

Для початку роботи з Selenium RC його потрібно завантажити і встановити. Сервер Selenium є звичайним JAR-файлом, який можна виконати в середовищі виконання (JRE або Java ™ Runtime Environment).

Після закінчення завантаження розпакуйте архів, що містить Selenium RC, і помістіть його куди-небудь на своїй файлової системи. Тепер можна запустити сервер Selenium, виконавши наступну команду:

```
java -jar selenium-server.jar
```

Встановивши модуль розширення, запустивши сервер, після чого ми приступили до роботи з web-додатком. Щоб відкрити Selenium IDE, натиснули в Firefox Tools> Selenium IDE. Коли Selenium IDE відкриється, натиснули Record. Selenium IDE почав запис: вона буде запам'ятовувати всі дії, які ми

виконуємо в браузері. Щоб увійти в тестовий додаток, виконуєио наступні кроки:

Увімкнувши запис в Selenium IDE переходимо на головну сторінку нашої системи, яка починається із форми авторизації користувача:

- вводимо ім'я користувача.
- вводима правильний пароль в поле Password.
- Натискаємо Login.

Після успішного входу в web-додаток повинна відкритися головна сторінка. На даний момент в тесті вже є кроки, проте ще немає ніяких перевірок того, що вони виконані успішно. Середовище Selenium повинно знати, на що їй треба подивитися, щоб зрозуміти, відобразилася чи сторінка так, як очікувалося.

Можна додати перевірочні дії, щоб переконатися, що додаток відображає правильний вміст. Не вимикаючи запис в Selenium, виконуємо наступні кроки: натискаємо правою кнопкою миші на будь-якому HTML-елементі, наприклад напису Your name і натискаємоніть verifyTextPresent Your name. Натискаємо Record, щоб зупинити запис. Якщо необхідно побачити свій тест в дії, натискаємо Run All (рисунок). Тест почне виконуватися і завершиться успішно.

Рис. 4.2. Запуск тестів - Run All

Щоб переконатися, що Selenium IDE насправді тестує наш додаток, відкриваємо IDE і поміняємо в коді значення імені користувача. Запускаємо програму і знову натискаємо Run All. На цей раз тест не пройде, так як web-програма не відобразить сторінку з правильними написами.

Експортуємо тест в JUnit. Ми записали наш тест і тепер можемо експортувати його для використання в JUnit. Вибераємо в Selenium IDE тест, натискаємо File> Export Test Case As> Java (JUnit) - Selenium RC. Вводимо

назву тесту (наприклад, IndexTests), і збережігаємо його, щоб потім імпортувати в Eclipse.

Виконуємо наступні кроки: Створюємо новий Java-проект, що включає в себе модульні тести Java, створені за допомогою JUnit. Додаємо тест JUnit.

Тепер додамо тести Selenium і JUnit. Для цього вибераємо в Package Explorer наш Java-проект і натисніть Build Path> Configure Build Path. На вкладці Libraries натискаємо Add JARs і вибераємо файл selenium-java-client-driver.jar. Коли вихідні Java-файли скомпілюємо, тест буде виглядати так, як показано в наступному лістингу

```
package com.example.mywebapp.tests;

import org.junit.Before;
import org.junit.Test;

import com.thoughtworks.selenium.SeleneseTestCase;

public class EnterInfoTests extends SeleneseTestCase {

    @Before
    public void setUp() throws Exception {
        setUp("http://localhost:8080/tested-webapp/index.jsp", "*firefox");
    }

    @Test
    public void testBadDate() {
        doLogin();
        selenium.type("name", "User");
        selenium.type("birthdate", "@#$#@");
        selenium.click("//input[@value='Submit']");
        selenium.waitForPageToLoad("30000");
        verifyTrue(selenium.isTextPresent("Please enter a valid date"));
    }

    @Test
    public void testValidDate() {
        doLogin();
        selenium.type("birthdate", "12/2/1999");
        selenium.click("//input[@value='Submit']");
        selenium.waitForPageToLoad("30000");
        verifyFalse(selenium.isTextPresent("Please enter a valid date"));
    }

    private void doLogin()
    {
        selenium.open("/tested-webapp/index.jsp");
        selenium.type("username", "user");
        selenium.type("password", "secret");
        selenium.click("//input[@value='Login']");
        selenium.waitForPageToLoad("30000");
    }
}
```

Після запуску сервера Selenium можна виконати модульний тест, натиснувши правою кнопкою миші на файл IndexTest.java і вибравши Run As> JUnit Test. Для того щоб сервер Selenium запустив екземпляр нашого браузера і виконав тести, може знадобитися деякий час. Після закінчення тесту ми отримали результат, що і для звичайних модульних тестів. На рисунку 4.3 представлено результати тестування модуля авторизації.

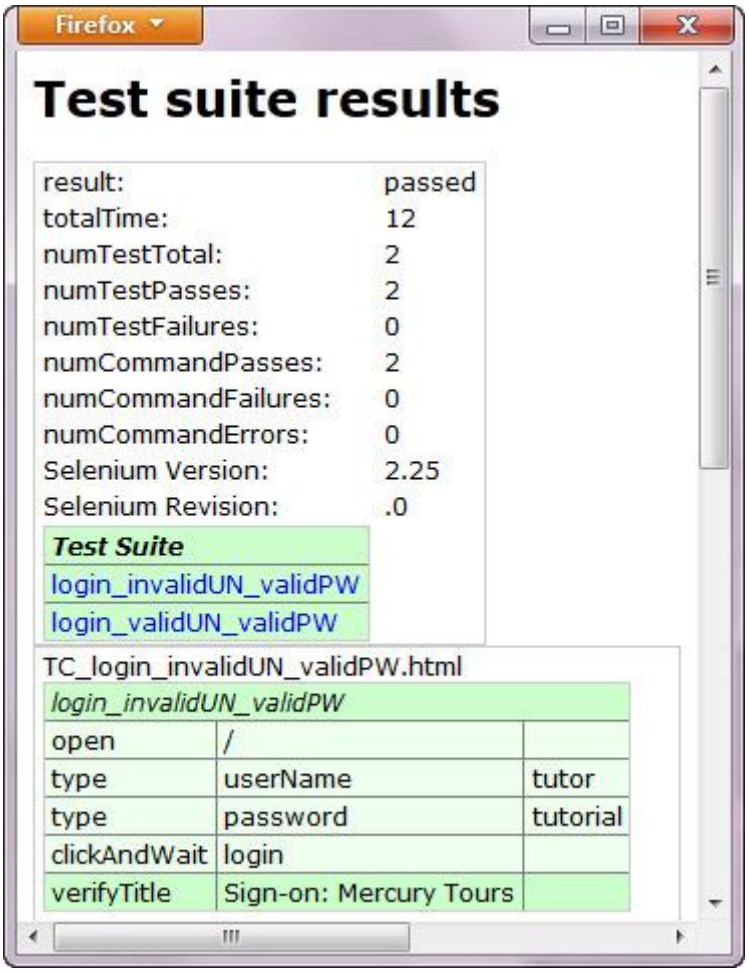


Рис. 4.3. Результати тестування модуля авторизації на сайті

4.2. Розгортання програмного продукту

В якості основи для побудови апаратної частини VDOM Box Server був обраний клас мініатюрних материнських плат формату mini-itx. Такі плати волоідють компактними розмірами, але в той же час забезпечують потрібну продуктивність для роботи web-сервера, який обслуговує сайти підприємств малого бізнесу. З урахуванням відносно невеликої ціни, підсумкове рішення виходить прийнятним за витратами.

Апаратна частина сервера складається з двох основних частин: Материнська плата формату mini-itx, на якій відбувається робота VDOM Box сервера і реалізується взаємодія з відвідувачами сайту. Електронна плата, яка відповідає за настройку системи і управління ліцензіями.

Вся програмна частина зберігається на зовнішній флеш-карті, доступній в системі тільки для читання. Таке рішення запобігає можливості «поломки» системи в результаті експлуатації. Також такий спосіб зберігання дозволяє дуже просто встановлювати систему на нові апаратні платформи і оновлювати програмне забезпечення.

Призначені для користувача дані зберігаються на жорсткому диску і можуть бути скопійовані на зовнішній носій за допомогою системи резервного копіювання. Ця система також дозволяє переносити web-додатки між різними пристроями. Електронна смарт-карта зберігає на собі дані про користувача і ліцензії. Вона визначає ліцензійні обмеження, такі, як максимальна кількість об’єктів, якф може бути створено в системі.

Програмна частина VDOM Box реалізована на платформно-незалежній мові Python, що забезпечило переносимість і простоту підтримки рішення. В якості операційної системи була обрана FreeBSD, так як вона забезпечує потрібну функціональність і невимоглива до ресурсів.

На сьогоднішній день версія технології VDOM існує в формі апаратно-програмного комплексу - пристрою, що виконує функцію web-сервера – VDOM Box Server (рисунок 4.4).

Рис. 4.4. Зовнішній вигляд VDOM Box Server

4.3. Інструкція користувача

Робота користувача з системою VBI Partners починається з авторизації. Користувач вводить свій логін і пароль. Якщо введення даних вірне, користувач благополучно заходить в систему, інакше він отримує повідомлення про помилку. Зовнішній вигляд сторінки Login зображений на рисунку 4.5.

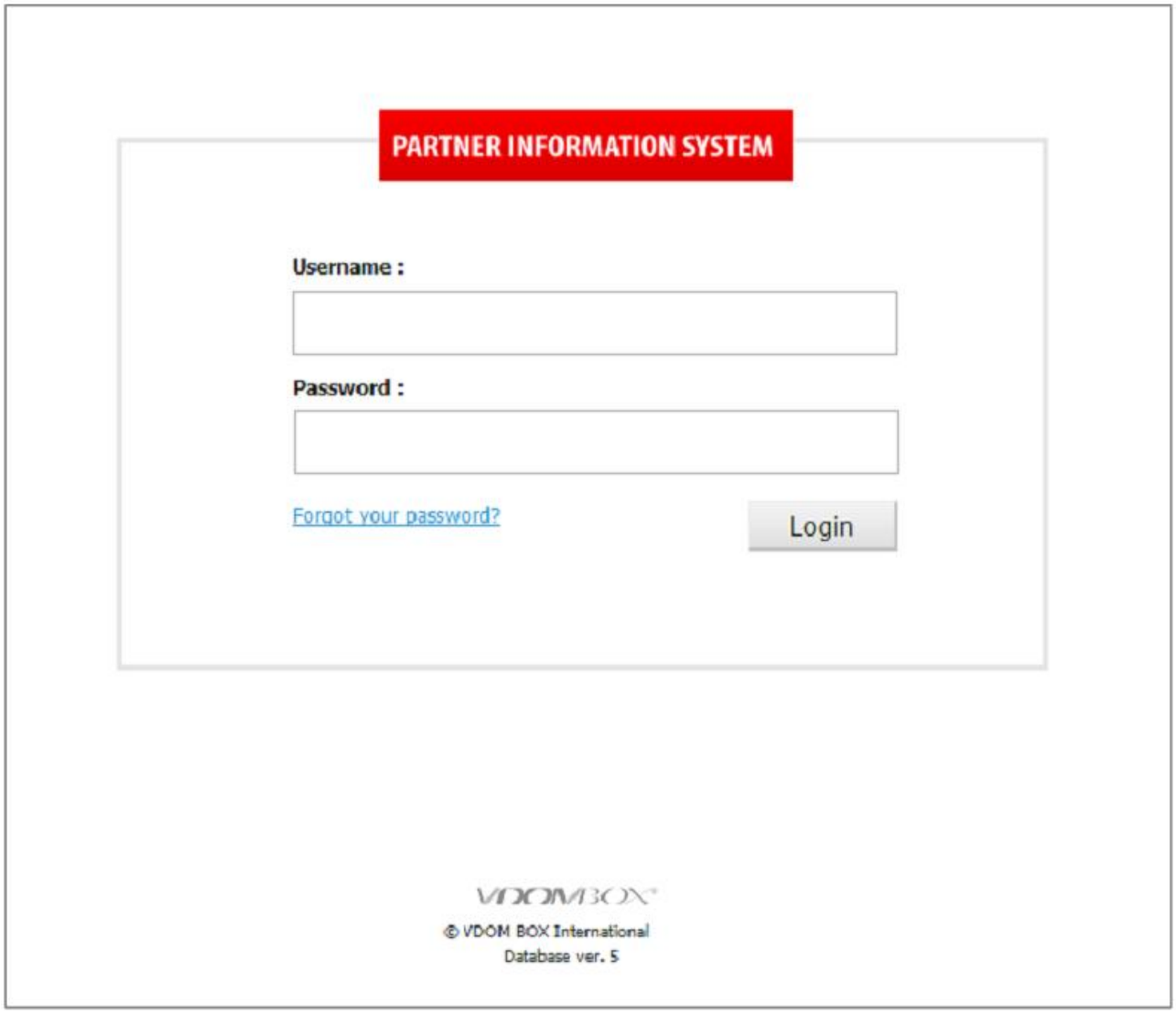


Рис. 4.5. Загальний вигляд сторінки для авторизації користувачів

Головна сторінка сайту. Справа зверху відображається інформація про партнера, від особи якого працює користувач. Зверху знаходиться рядок меню, що дозволяє перейти в розділ, що цікавить. Основну частину сторінки займають дві інформаційних панелі: Orders і Parts.

Панель Orders відображає список останніх замовлень. У списку вказується опис замовлення, дата його створення, а також поточний стан. Вгорі панелі знаходиться кнопка, для переходу до повного списку замовлень.

Панель Parts дозволяє виконувати пошук по конкретних товарах. Враховуються товари, які коли-небудь проходили через склади поточного партнера. Результати пошуку відображаються на дисплеї, з’явиться

повідомлення на окремій сторінці - Search Parts. Зовнішній вигляд сторінки Dashboard зображений на рисунку 4.6.

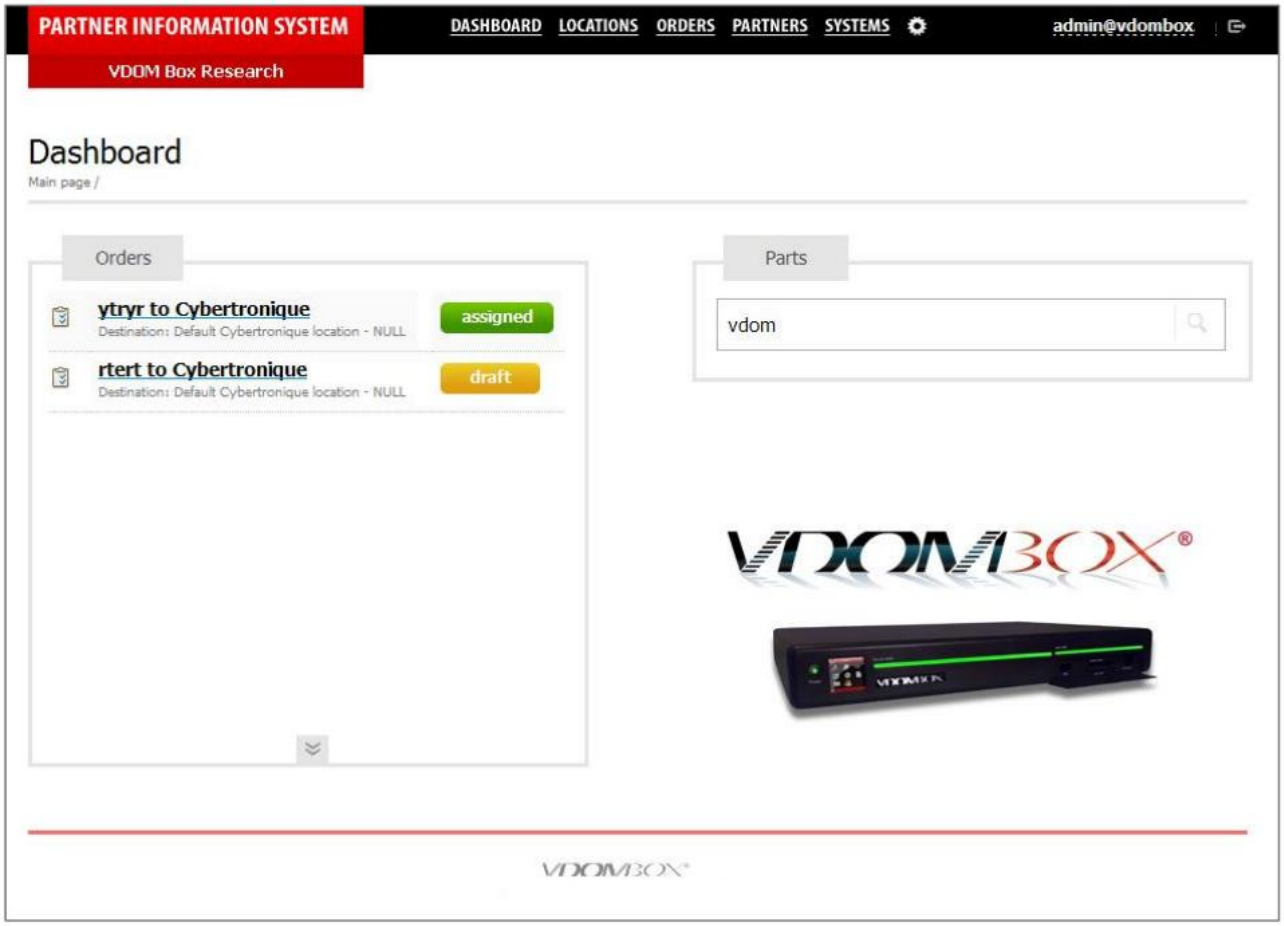


Рис. 4.6. Загальний вигляд головної сторінки

Сторінка Location List. Сторінка відображає список складів, що належать поточному партнеру. Для кожного складу наводиться його назва, а також інформація про його місце розташування. На даній сторінці користувач може:

- Подивитися детальну інформацію про конкретний складі. Для цього необхідно перейти за посиланням Location Details ->.

- Додати новий склад. Для цього необхідно клацнути по кнопці Add location. Зовнішній вигляд сторінки Location List зображений на рисунку 4.7.

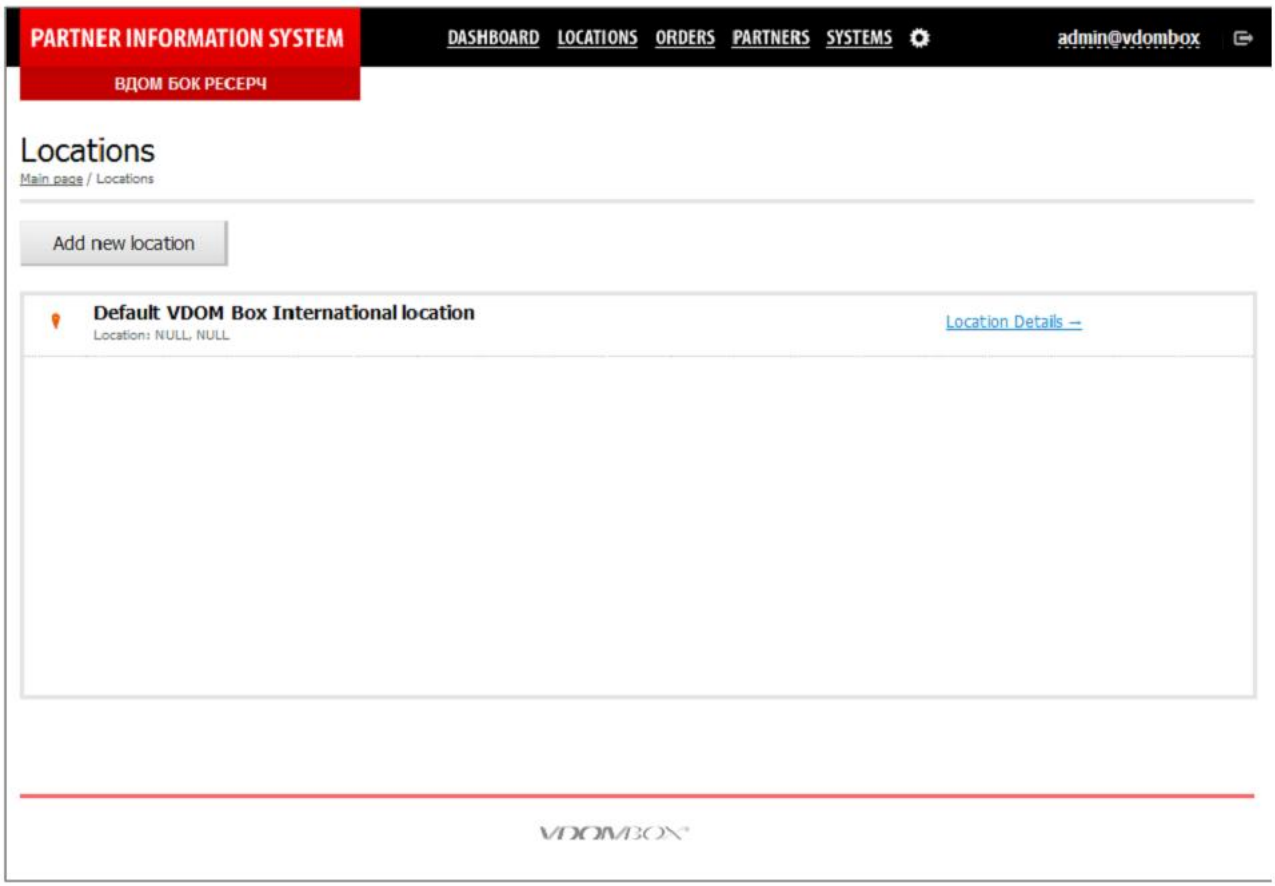


Рис. 4.7. Сторінка Location List

Сторінка Location Details. Сторінка відображає детальну інформацію по конкретному складу. Зліва наводиться інформація про назву, опис та повну адресу складу. Справа показано зведення за доступними на даному складі товарами (найменування і кількість). Зверху розташовані кнопки для виконання специфічних дій. На даній сторінці користувач може:

- Перейти до додавання нового товару, натиснувши кнопку Add part.

- Перейти до додавання «готового рішення», натиснувши кнопку Add Package.

- Відредагувати інформацію про склад, натиснувши кнопку Edit location details.

Подивитися список товарів, певного типу (наприклад, всі флеш-карти, що лежать на складі), перейшовши за посиланням Details у відповідній колонці. Зовнішній вигляд сторінки Location Details зображений на рисунку 4.8.

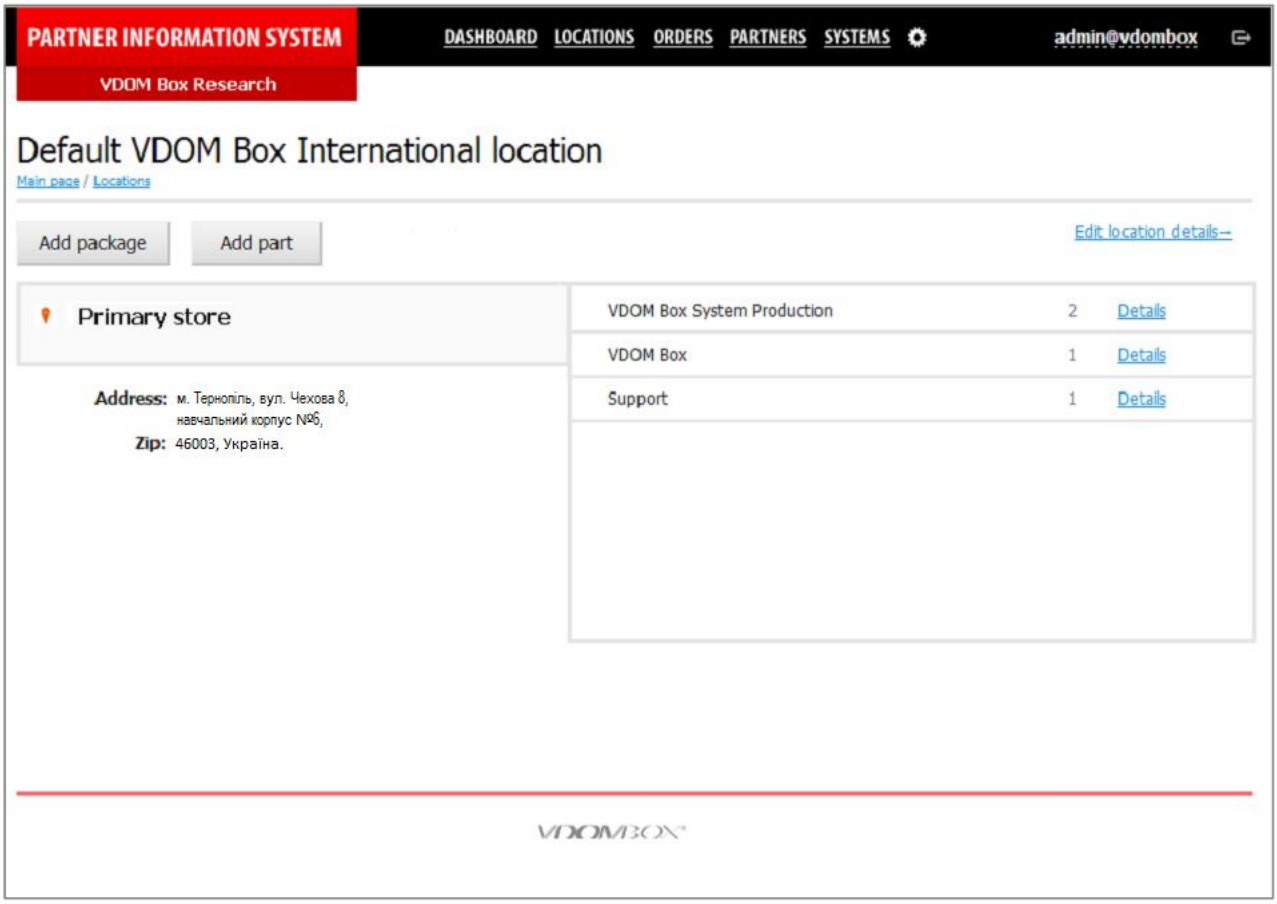


Рис. 4.8. Загальний вигляд сторінки Location Details

Сторінка Edit Location. На даній сторінці виконується додавання нового або редагування вже існуючого складу поточного партнера. Перед створенням складу необхідно вказати інформацію про нього. Поля, помічені зірочкою, обов’язкові для заповнення. За натисканняv кнопки Create, в разі, якщо вказана вся необхідна інформація, відбувається створення (зміна) складу і користувач перенаправляється на сторінку Location List.

Якщо не всі обов’язкові поля були заповнені, користувач отримує про це повідомлення. Натискання кнопки Back також переводить користувача на сторінку Location List. Зовнішній вигляд сторінки Edit Location зображений на рисунку 4.9.

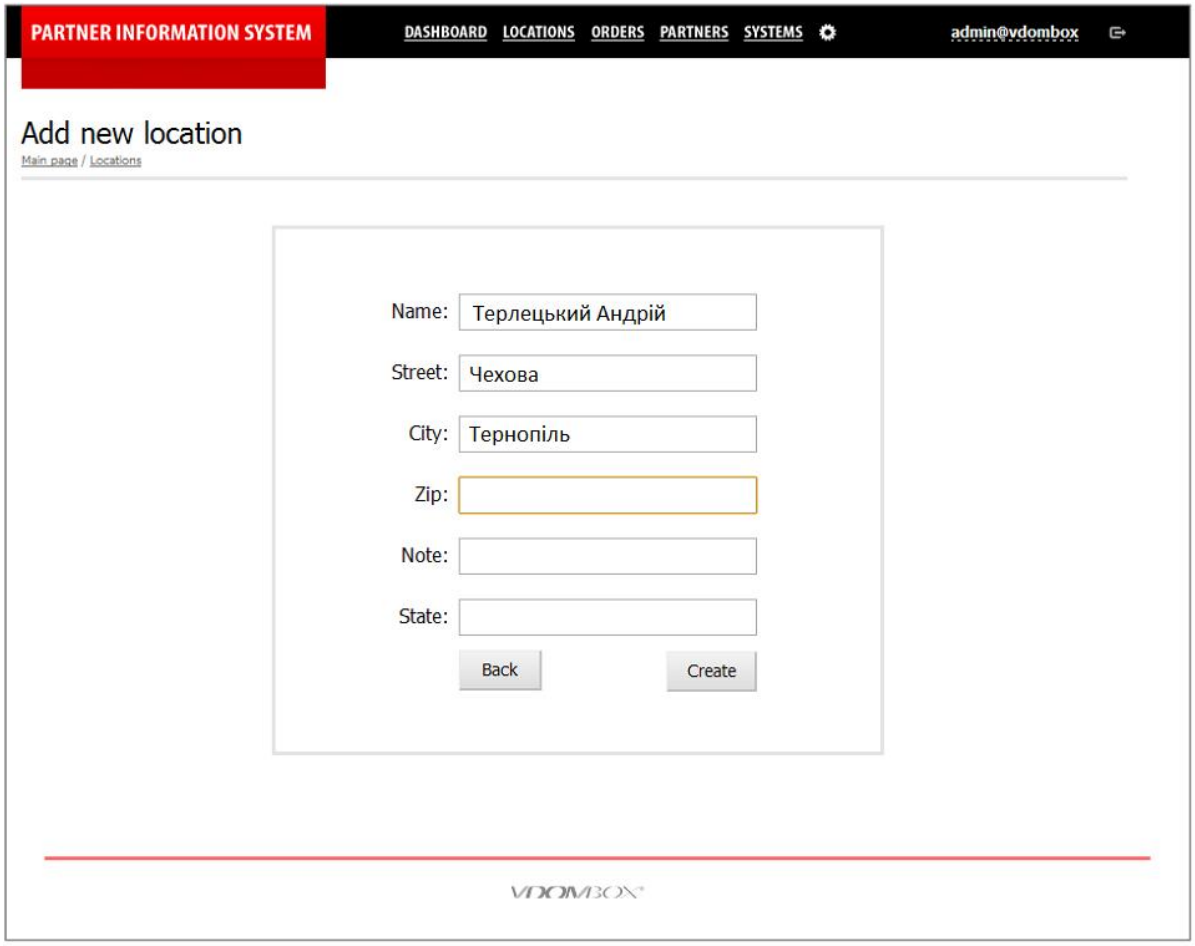


Рис. 4.9. Загальний вигляд сторінки Edit Location

Сторінка Order List. Сторінка показує список вхідних і вихідних замовлень поточного партнера. Для кожного замовлення вказується наступна інформація: опис, ім’я партнера для якого виконується замовлення, пункт призначення, дата створення, поточний стан. Сторінка може працювати в двох режимах: оперативному та архівному. В оперативному режимі відображаються «активні» замовлення, тобто такі, стан яких належить множині {DRAFT, CONFIRMED, ASSIGNED, SENT}. Інформація про неактивні замовлення (Стану CANCELLED і DELIVERED) відображається в архівному режимі.

На сторінці можна:

- Подивитися детальну інформацію по конкретних замовленнях, клікнувши на його імені (Виконається перехід на сторінку Order Details).

- Змінити режим відображення сторінки (оперативний/архівний). Зовнішній вигляд сторінки Order List зображений на рисунку 4.10.

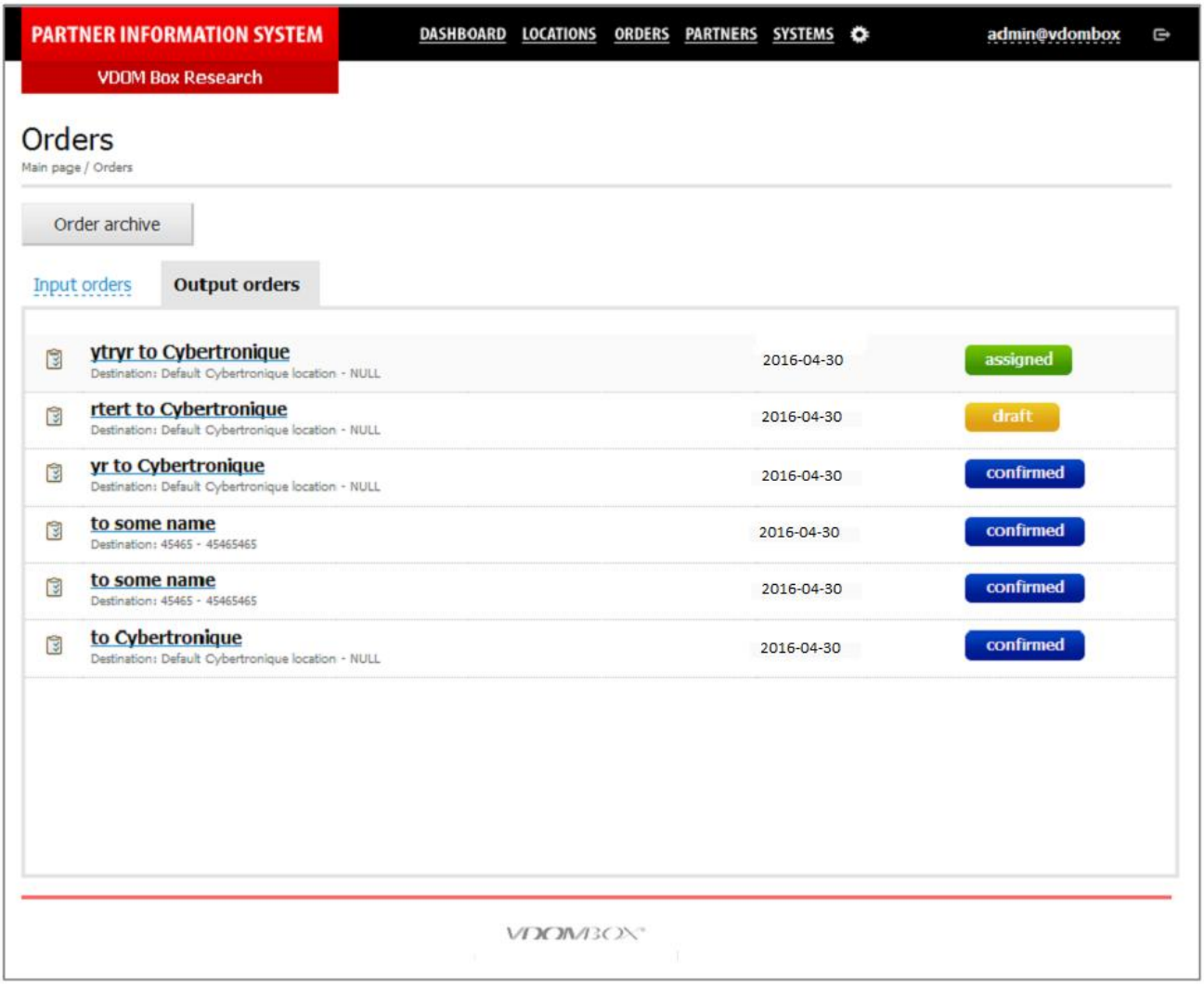


Рис. 4.10. Загальний вигляд сторінки Order List

Сторінка Order Details. Сторінка відображає дані про конкретні замовлення. Зліва вказані опис замовлення, назви і адреси пункту відправлення та пункту призначення, а також поточний стан замовлення. Справа показаний список позицій замовлення (найменування товару, кількість). На даній сторінці користувач може:

- Перейти до редагування замовлення (загальної інформації, а також інформації про позиції замовлення), натиснувши кнопку Edit Order.

- Переглянути історію зміни станів замовлення, натиснувши кнопку History. Історія замовлення буде показана у спливаючому вікні.

- Змінити стан замовлення, вибравши один з пунктів, запропонованих внизу праворуч.

Логіка зміни стану залежить від того, в якому стані поточне замовлення знаходиться в даний момент, а також який статус має дане замовлення для поточного партнера (вхідний або вихідний).

Логіка зміни стану замовлення в залежності від поточного стану:

- DRAFT. Можна скасувати замовлення, тобто перевести його в стан CANCELLED вибравши пункт Cancel Order. Можна виконати підтвердження замовлення (стан CONFIRMED), вибравши пункт Confirm Order. Дані дії можуть виконувати обидва партнера (постачальник і споживач).

- CONFIRMED. Можна приготувати замовлення до відправки (перевести в стан ASSIGNED). Для цього необхідно вибрати пункт Assign Order і потрапити на сторінку Assign Order, на якій потрібно буде додати в замовлення конкретні товари.

Після додавання всіх необхідних товарів (тобто повністю покривають позиції замовлення), замовлення автоматично буде переведено в стан ASSIGNED. Дану дія може здійснювати тільки постачальник.

- ASSIGNED. Можна помітити замовлення як відправлене (перевести в стан SENT). Дану дія може здійснювати тільки постачальник.

- SENT. Замовлення можна помітити як доставлене (перевести в стан DELIVERED). Дану дія може здійснювати тільки споживач. Зовнішній вигляд сторінки Order Details зображено на рисунку 4.11.

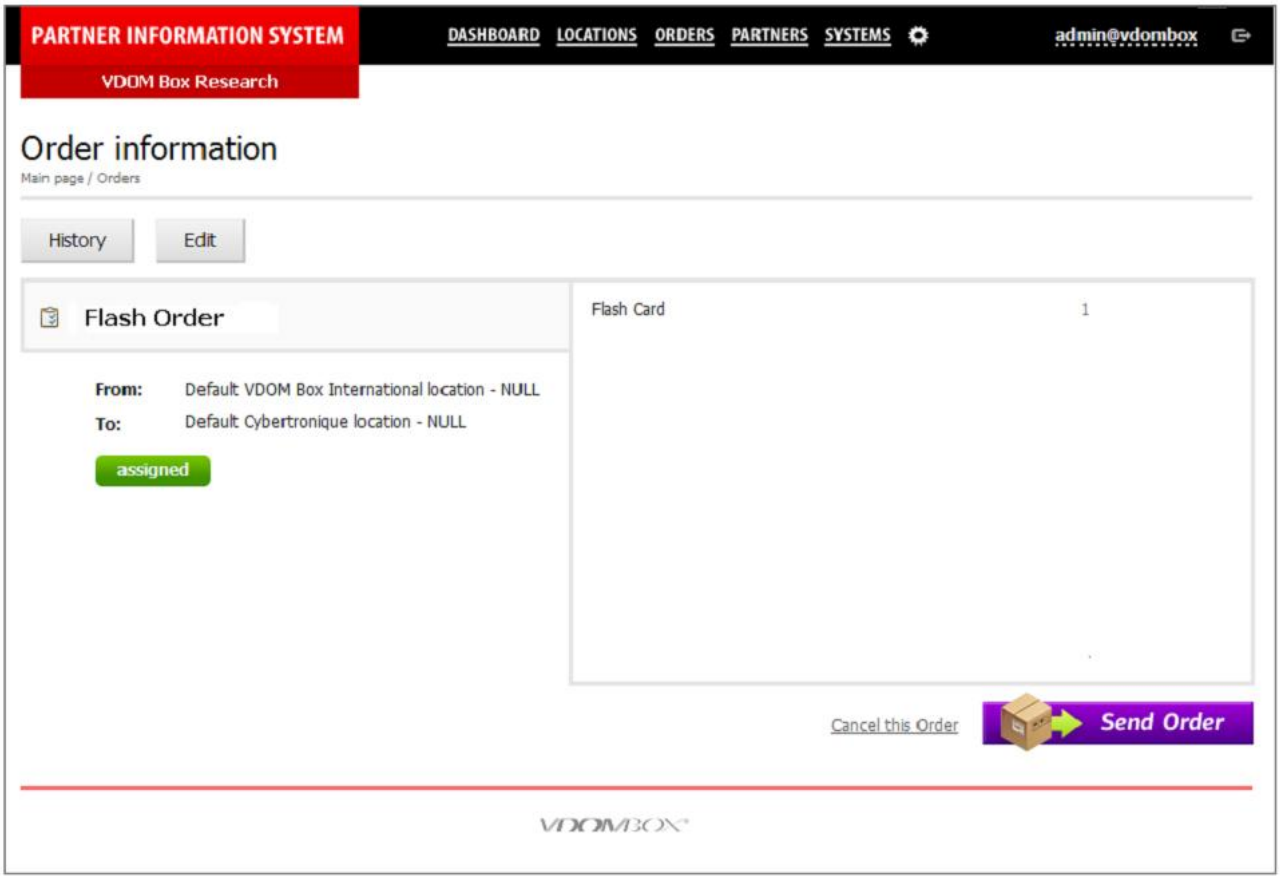


Рис. 4.11. Загальний вигляд сторінки Order Details

Сторінка Edit Order. Сторінка призначена для редагування існуючого замовлення, або для створення нового. У верхній частині задається загальна інформація про замовлення: опис, пункт відправлення замовлення, партнер, з яким відправляється замовлення, а також пункт призначення. Нижня частина призначена для зміни позицій замовлення. У лівому списку відображаються найменування доступних товарів. Натискання по елементу списку призводить до додавання відповідного типу товару в замовлення. У правому списку відображаються позиції, вже додані в замовлення. Користувач може змінити кількість товару, ввівши число у відповідне поле введення. Видалення конкретної позиції здійснюється натисканням по «хрестику» у відповідному рядку. По закінченню редагування замовлення користувач натискає кнопку Save changes, і якщо всі необхідні дані вказані, виконується збереження даних про замовлення, а користувач потрапляє на сторінку Order Details. Зовнішній вигляд сторінки Edit Order зображено на рисунку 4.12.

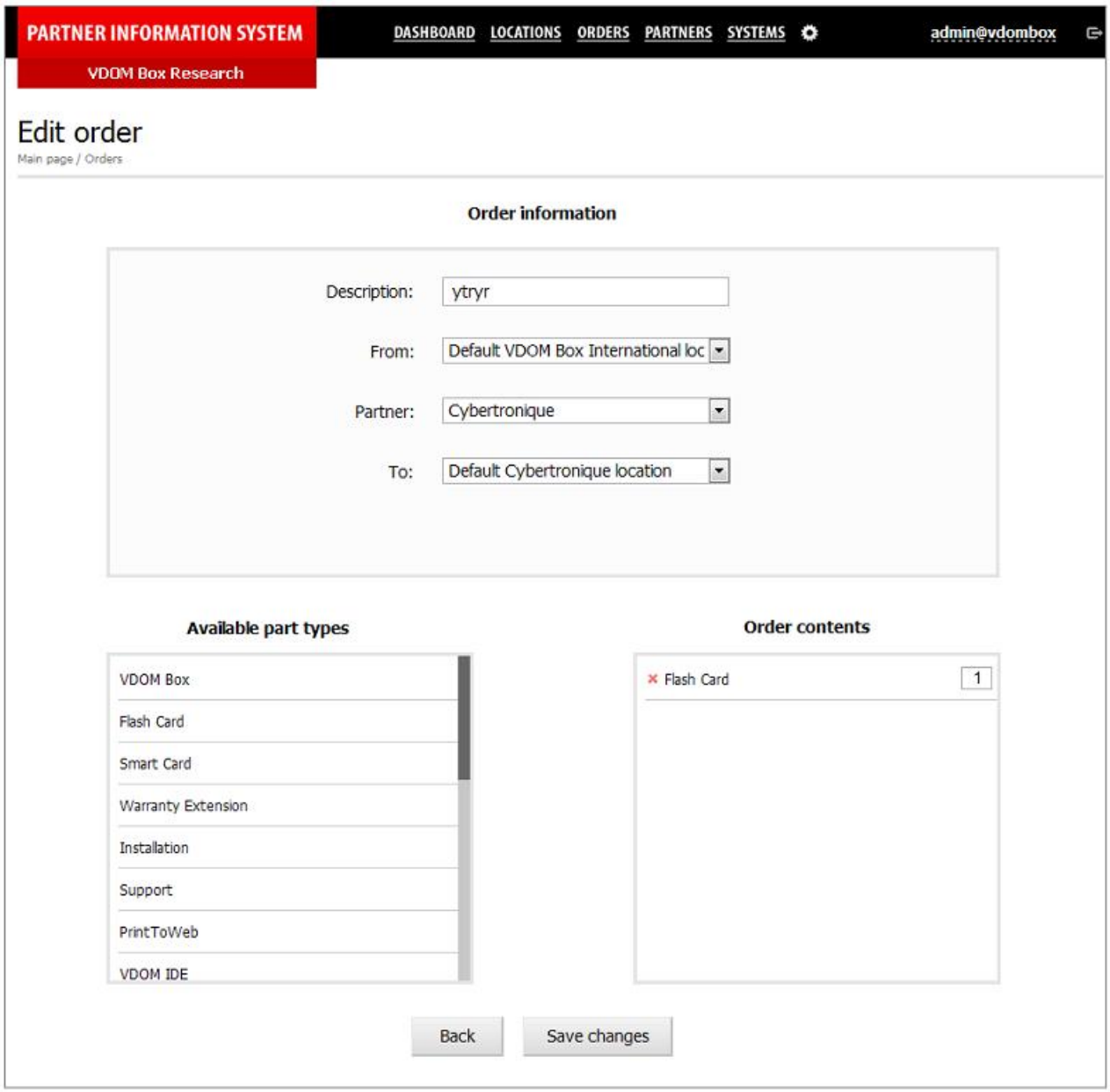


Рис. 4.12. Загальний вигляд сторінки Edit Order

Сторінка Assign Order. Дана сторінка використовується для додавання в замовлення конкретних товарів. Розглянемо верхню частину сторінки. Зліва відображається інформація про пункти відправки і призначення. Справа показані позиції замовлення і їх заповненість (виходячи з того, які товари вже додані в замовлення).

Нижня частина використовується безпосередньо для додавання товарів в замовлення. В лівому списку відображаються доступні на складі товари, праворуч - товари, додані в замовлення. Додавання/видалення товарів з

відповідних списків здійснюється натисканням на відповідні елементи цих списків. Натискання на кнопку Save призводить до збереження зроблених змін і переходу на сторінку Order Details. Якщо користувач заповнив усі позиції замовлення, то виконується перевід його стану в ASSIGNED. Зовнішній вигляд сторінки Assign Order зображено на рисунку 4.13.

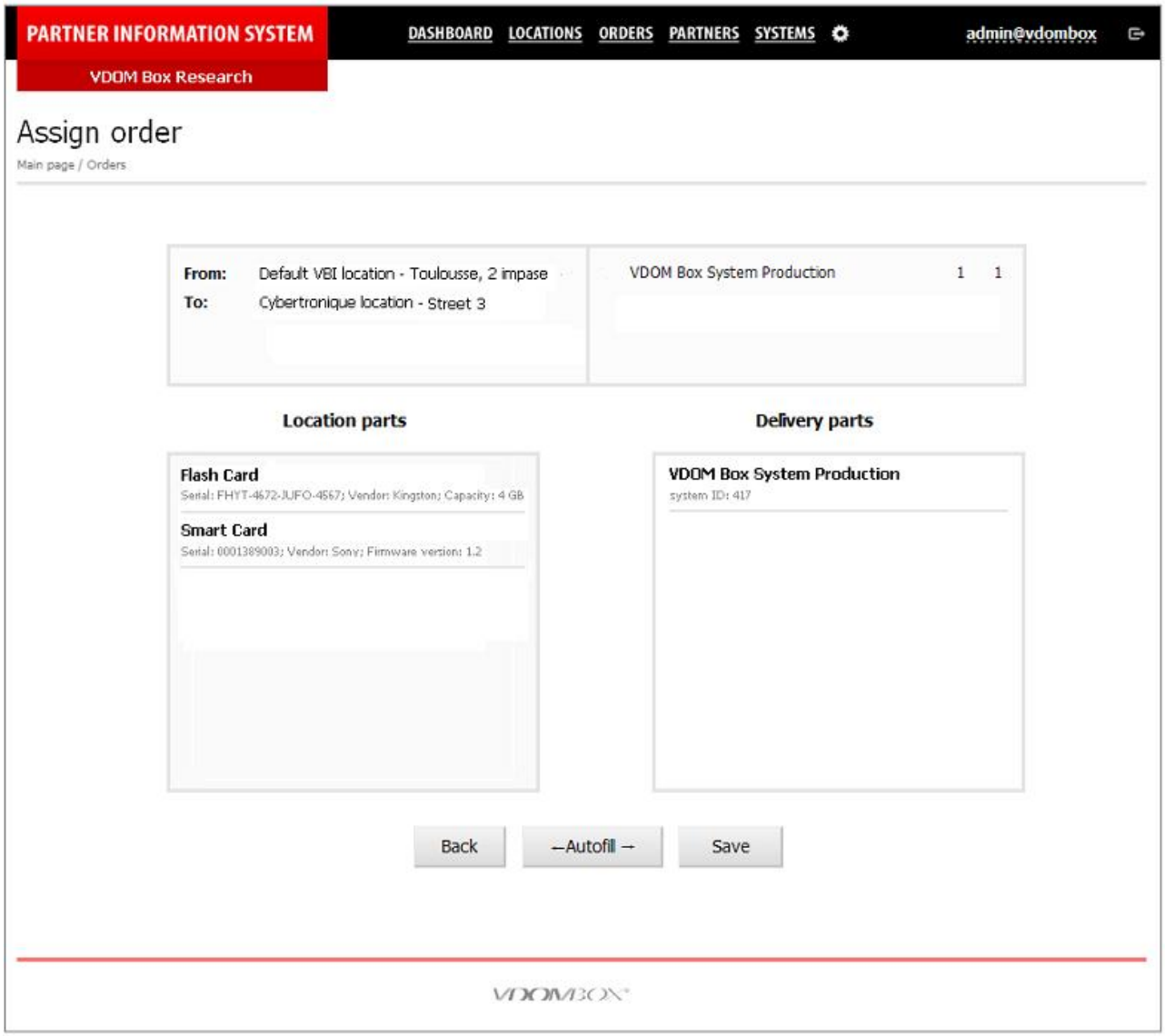


Рис. 4.13. Загальний вигляд сторінки Assign Order

Сторінка Partner List. Сторінка відображає список партнерів, з якими працює поточний партнер. Всі партнери згруповані за відповідними списками: постачальники, споживачі (дистриб’ютори) і клієнти. Для кожного партнера вказується його назва (ім’я) і спеціальний код.

На сторінці можна:

- Створити замовлення, пов’язане з одним з партнерів, натиснувши кнопку New order.

- Додати одного з існуючих партнерів в список своїх постачальників/споживачів/клієнтів.

- Видалити партнера зі списку своїх постачальників / споживачів / клієнтів.

- Подивитися дані про партнера.

- Створити нового партнера.

Сторінка Partner Details. Сторінка відображає докладні відомості про конкретних партнерів. Дані відомості включають в себе:

- Назву розглянутого партнера і його спеціальний код.

- Список співробітників (користувачів системи) партнера.

- Журнал вхідних замовлень розглянутого партнера, виконуваних поточним партнером.

- Список вихідних замовлень розглянутого партнера, зроблених поточним партнером.

- Список систем (готових рішень), проданих поточним партнером.

Якщо проглядається поточний партнер (тобто партнер, від імені якого працює поточний користувач), то для нього можна редагувати дані про нього самого і про його співробітників.

Сторінка Part List. Сторінка показує список товарів певного типу, розташованих на конкретному складі. Для кожного товару вказується інформація про всі його атрибути. Значення атрибутів, можна не тільки подивитися, але і також відредагувати. Зміни значень атрибутів вступають в силу після натискання кнопки Update Parts. Натискання кнопки Back призводить до повернення користувача на сторінку Location Details. Зовнішній вигляд сторінки Part List зображений на рисунку 4.14.

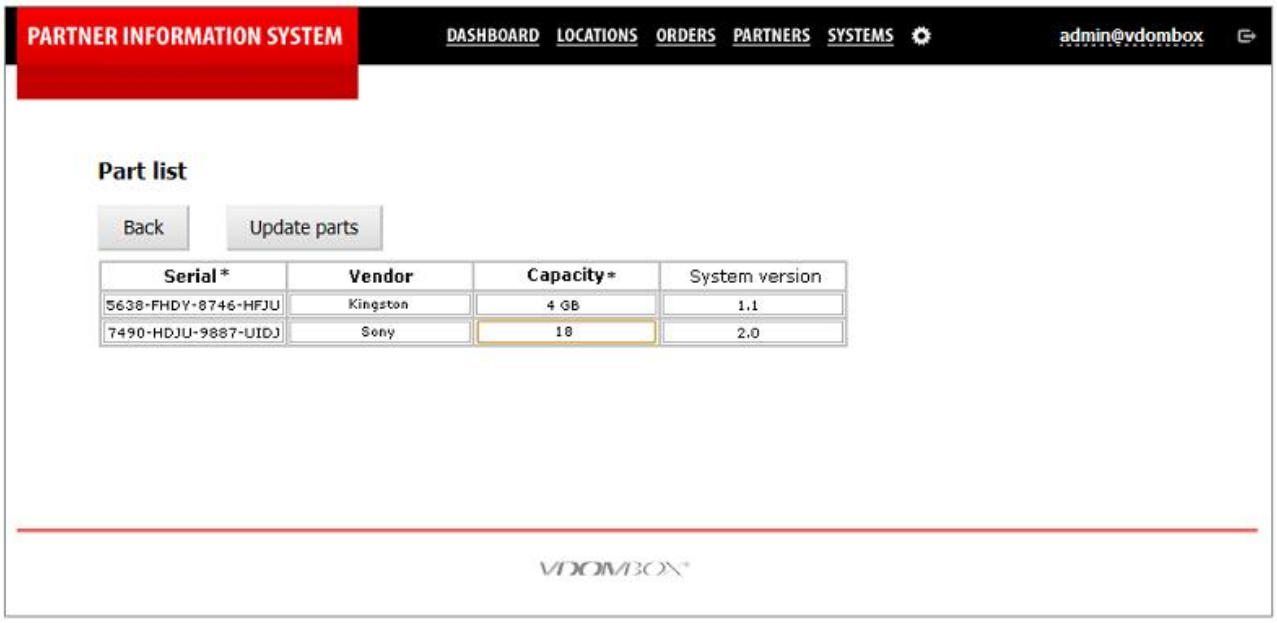


Рис. 4.14. Загальний вигляд сторінки Part List

Сторінка Part Details. Сторінка відображає докладні відомості про конкретний товар. Зліва представлено найменування товару і список його атрибутів. Також тут наводяться всі складові частини даного товару (якщо він є складовим). Справа представлений список всіх замовлень, в яких «представлений» даний товар. Даний список наводиться в хронологічному порядку (тобто в порядку доставки відповідних замовлень).

Сторінка Search Parts. Сторінка дозволяє здійснювати пошук по товарах. Містить стандартне поле введення пошукового рядка і кнопку пошуку. Результати пошуку видаються у вигляді списку товарів, для кожного з яких наводиться найменування і набір ключових атрибутів. Пошук ведеться по підрядку за найменуваннями товарів і по їх атрибутах. Пошук здійснюється серед тих товарів, які коли-небудь проходили через поточного партнера.

Сторінка Add Part. Інтерфейс сторінки дозволяє додати нову одиницю товару на конкретний склад. У верхній частині панелі через список, що випадає вказується найменування товару. На панелі атрибутів користувачеві пропонується заповнити значення атрибутів товару. Атрибути, відмічені зірочкою обов’язкові для заповнення. Крім того для прискорення процесу введення даних значення деяких атрибутів вже заповнені значеннями за

замовчуванням. Для здійснення операції додавання товару необхідно натиснути кнопку Add. Якщо всі обов'язкові значення введені, товар успішно додається на склад, а користувач перенаправляється на сторінку Location Details. В іншому випадку користувачеві виводиться повідомлення про помилку і пропозицію заповнення обов'язкових полів. Натискання кнопки Back повертає користувача на сторінку Location Details. Зовнішній вигляд сторінки Add Part зображений на рисунку 4.15.

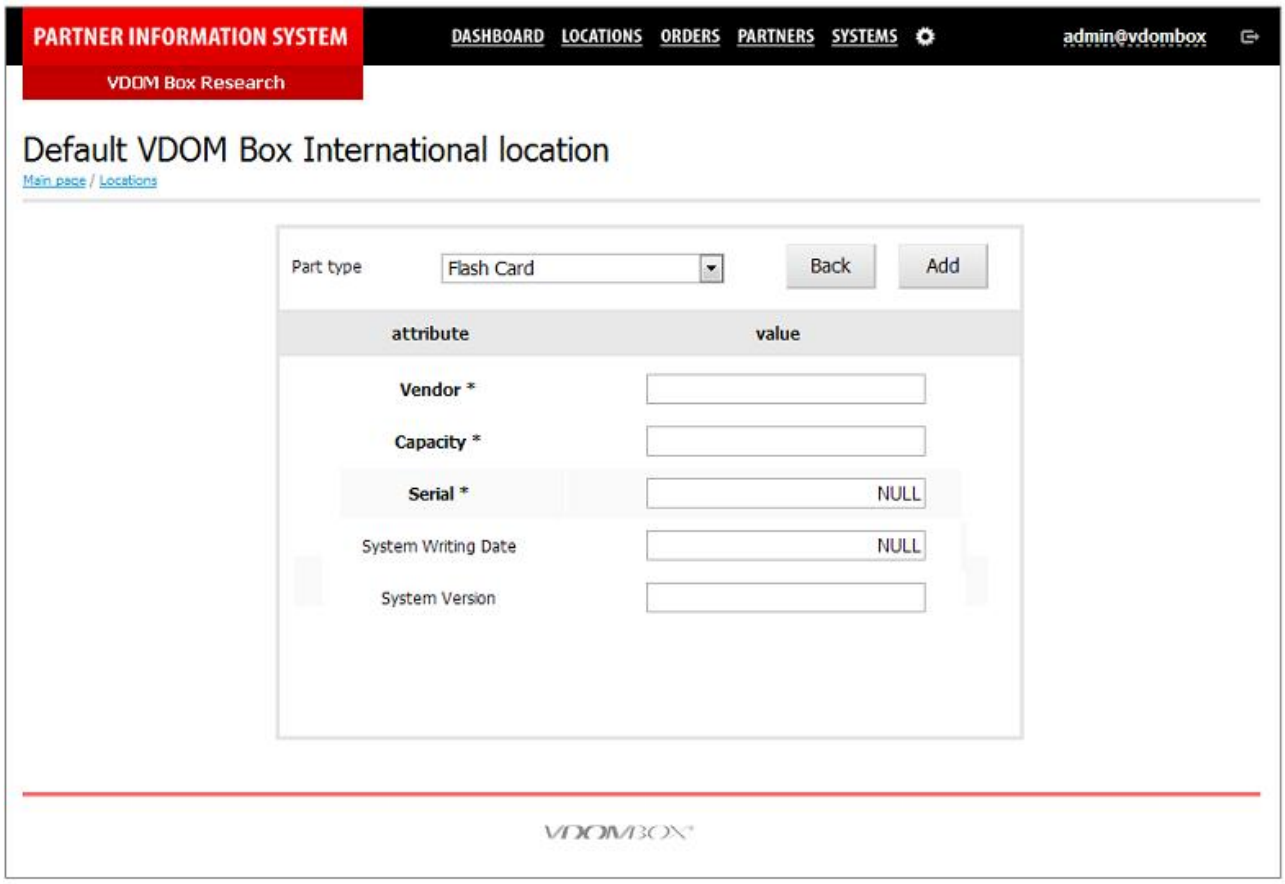


Рис. 4.15. Загальний вигляд сторінки Add Part

Висновки до розділу 4

Здійснено опис процедур тестування та їхніх результатів, описані тест-вимоги до програмного забезпечення, а також виявлені дефекти. Розкрито питання встановлення та налаштування програмного забезпечення на сервері, а також вказані вимоги, дотримання яких необхідно для користування системою, описана інструкція користувача для роботи із системою.

ВИСНОВКИ

В результаті проведеної роботи була реалізована система підтримки продажів VBI Partners, що відповідає вимогам замовника. Були розроблені реляційна база даних, бізнес-шар логіки, зовнішній API для інтеграції зі сторонніми додатками, а також призначений для користувача веб-інтерфейс. В даний час система тестується підрозділами компанії, а також її партнерами, що займаються розповсюдженням продукції компанії

Завдяки використанню універсального механізму платформа VDOM сервер дозволила використати наступні перевагами в ході реалізації описаного в роботі проекту:

- Простота і гнучкість використання.
- Локалізація будь-яких об'єктно-орієнтованих веб-додатків.
- Підтримка різних мов і можливість динамічної зміни мов інтерфейсу.

На основі концепцій V.D.O.M. був використаний новий підхід до створення Інтернет-додатків. Модель об’єктів, яка отримала розвиток "компонентною моделлю", спосіб побудови додатків, який дозволяє значно спростити процес розробки додатків для бізнесу.

Компонентно-орієнтовані додатки мають велику перспективу в комерційних розробках малої і середньої величини. Оскільки сфера діяльності багатьох організацій інформаційної галузі нашої країни найчастіше пов'язана з виконанням малобюджетних проектів, в яких, в силу жорсткої конкуренції, немає часу на створення інфраструктури додатків, виникає особлива потреба в скороченні витрат проектування і реалізації.

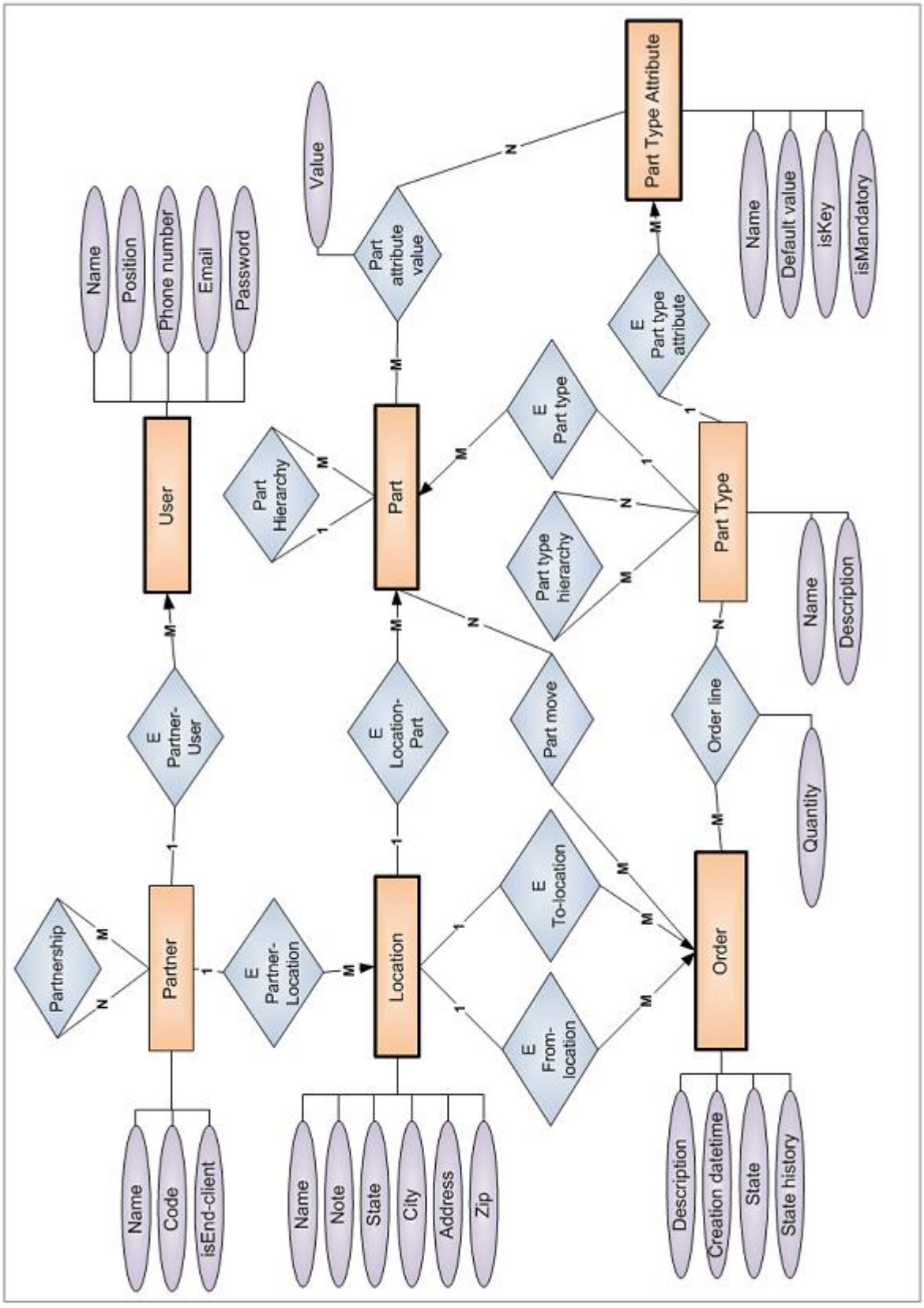
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бабанов А.М. – Технология разработки программного обеспечения: структурный подход. – Томск: Изд-во НТЛ, 2006. – 220 с Буров Є. В.
2. Комп’ютерні мережі: підручник / Євген Вікторович Буров. — Львів: «Магнолія 2006», 2010. — 262 с. ISBN 966-8340-69-8
3. Korboulewsky N. V.D.O.M. BOX. Technology V.D.O.M. / V.D.O.M. v2 – W.H.O.L.E. v1- EI.V.D.O.M v1. – 7 rue Saint Henri 31000 Toulouse, 2007. – P. 76
4. Ландэ Д. В., Снарский А. А., Безсуднов И. В. Интернетика: Навигация в сложных сетях: модели и алгоритмы. — М.:Либроком (Editorial URSS), 2009. — 264 с. ISBN=978-5-397-00497-8.
5. Fowler M. Patterns of Enterprise Application Architecture. – Addison Wesley, 2004. –P. 523
6. Дейт К.Дж. Введение в системы баз данных. – Вильямс, 2006. 1328 с.
7. Инженерное программирование для проектирования программного обеспечения. -М.: Радио і связь, 1985, -512с.
8. Бойков.В., Савинков В.М. Проектирование баз данных информационных систем. М. Мир 1997
9. Бердтис А. Структуры данних. - М.: Статистика, 1974, - 408 с.
10. Горбань О.М., Бахрушин В.Є. Основі теорії систем та системного аналізу. - Запоріжжя, ГУ "ЗІДМУ", 2004, ISBN 966-8227-23-9
11. Майо Д. Самоучитель Microsoft Visual Studio 2010 = Microsoft Visual Studio 2010: A Beginner’s Guide (A Beginners Guide). — С.: «БХВ-Петербург», 2010. — С. 464. — ISBN 978-5-9775-0609-0
12. Алекс Макки Введение в .NET 4.0 и Visual Studio 2010 для профессионалов = Introducing .NET 4.0: with Visual Studio 2010. — М.: «Вильямс», 2010. — С. 416. — ISBN 978-5-8459-1639-6

13. Кен Хендерсон Професійне керівництво з SQL Server: структура та реалізація. — М.: Издательский дом «Вильямс», 2006. — С. 1056. ISBN 5-8459-0912-0
14. Чураков Михаил. Муравьиные алгоритмы [Електронний ресурс] / Михаил Чураков,Андрей Якушев. // Режим доступу: http://rain.ifmo.ru/cat/data/theory/unsorted/ant-algo-2006/article.pdf.
15. Бормашов Д. А. “Кластерный анализ текстов”: Дипломная работа [Електронний ресурс] /Д. А. Бормашов. Режим доступу: http://inf.tsu.ru/library/DiplomaWorks/CompScience/2006/bormashov/diplom.pdf.
16. Чубукова И.А. Data Mining БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2006.
17. Дюк В.А., Самойленко А.П. Data Mining: учебный курс. – СПб.: Питер, 2001.
18. Барсегян А. А., Куприянов М. С., Степаненко В. В., Холод И. И. Методы и модели анализа данных: OLAP и Data Mining. – СПб.: БХВ-Петербург, 2004. – 336 с.
19. Дивак М. П., Шпінталь М.Я., Козак О.Л., Струбицька І.П., Спільчук В.М., Піговський Ю.Р. Методичні рекомендації до виконання дипломної роботи освітньо кваліфікаційного рівня «бакалавр» студентам усіх форм навчання для напряму підготовки 6.050103 – «Програмна інженерія» // Тернопіль : ФОП Шпак П.П. - 2014. - 54 с.

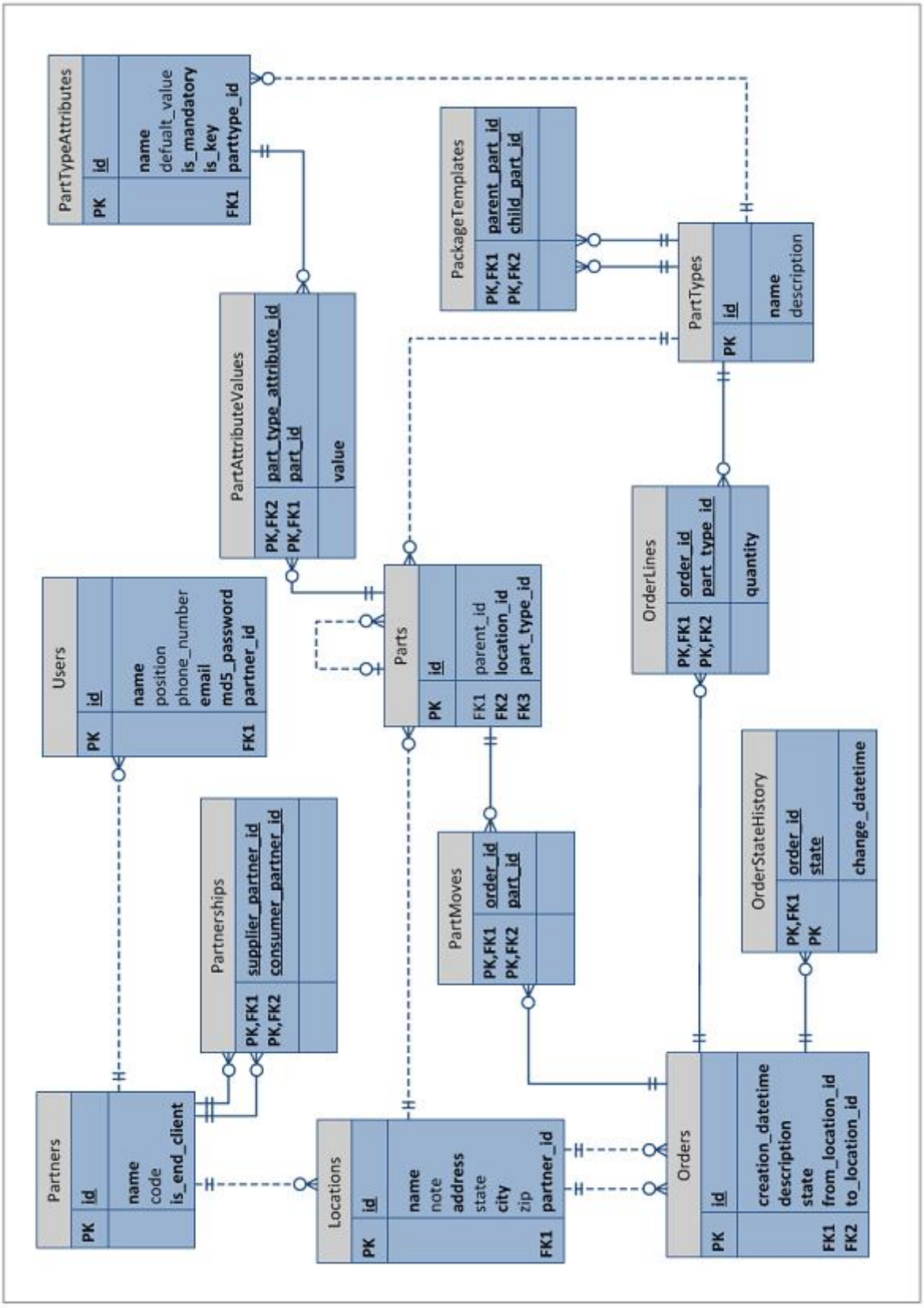
ДОДАТОК А

ER-ДІАГРАМА



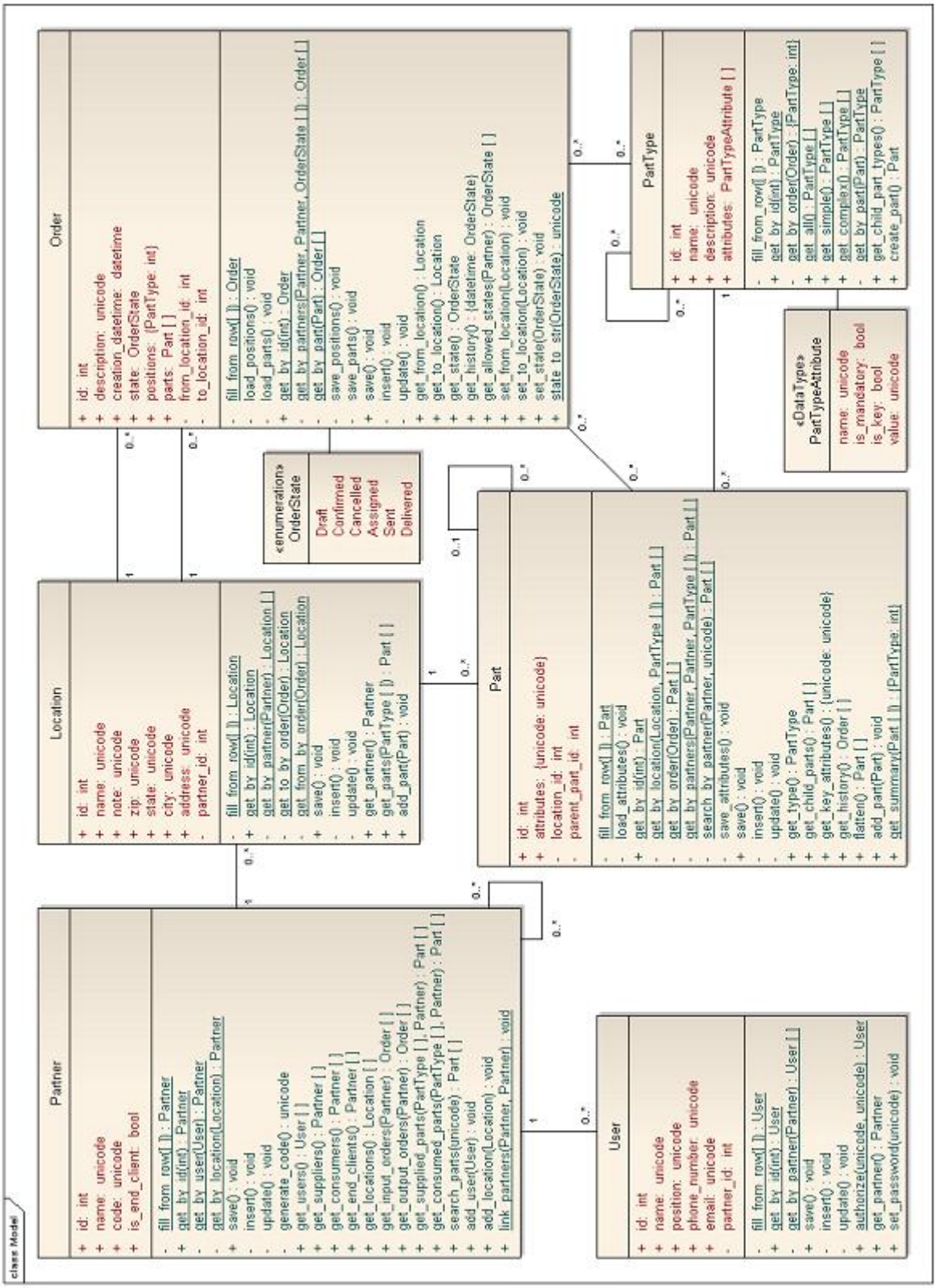
ДОДАТОК Б

РЕЛЯЦІЙНА ДІАГРАМА



ДОДАТОК В

ДІАГРАМА КЛАСІВ ШАРУ БІЗНЕС-ЛОГІКИ



ДОДАТОК Г

ЛІСТИНГ ОСНОВНИХ МОДУЛІВ СИСТЕМИ

```
#МОДУЛЬ УПРАВЛІННЯ КЛІЄНТАМИ
from __future__
import
unicode_literals
from future.builtins import str, zip
from collections import defaultdict, OrderedDict
from datetime import datetime, timedelta
from django.db.models import Manager, Q
from django.utils.timezone import now
from mezzanine.conf import settings
from mezzanine.core.managers import CurrentSiteManager
class CartManager(Manager):
    def from_request(self, request):
        """
        Return a cart by ID stored in the session, updating its last_updated
        value and removing old carts. A new cart will be created (but not
        persisted in the database) if the session cart is expired or missing.
        """
        cart_id = request.session.get("cart", None)
        cart = self.current().filter(id=cart_id)
        last_updated = now()
        # Update timestamp and clear out old carts.
        if cart_id and cart.update(last_updated=last_updated):
            self.expired().delete()
        elif cart_id:
            # Cart has expired. Delete the cart id and
            # forget what checkout step we were up to.
            del request.session["cart"]
            cart_id = None
            try:
                del request.session["order"]["step"]
            except KeyError:
                pass
        # This is a cheeky way to save a database call: since Cart only has
        # two fields and we know both of their values, we can simply create
        # a cart instance without taking a trip to the database via the ORM.
        return self.model(id=cart_id, last_updated=last_updated)
    def expiry_time(self):
        """
        Datetime for expired carts.
        """
        return now() - timedelta(minutes=settings.SHOP_CART_EXPIRY_MINUTES)
    def current(self):
        """
        Unexpired carts.
        """
        return self.filter(last_updated__gte=self.expiry_time())
    def expired(self):
        """
        Expired carts.
        """
        return self.filter(last_updated__lt=self.expiry_time())
class OrderManager(CurrentSiteManager):
    def from_request(self, request):
        """
        Returns the last order made by session key. Used for
        Google Anayltics order tracking in the order complete view,
        and in tests.
        """
        orders = self.filter(key=request.session.session_key).order_by("-id")
        if orders:
```

```
            return orders[0]
        raise self.model.DoesNotExist
    def get_for_user(self, order_id, request):
        """
        Used for retrieving a single order, ensuring the user in
        the given request object can access it.
        """
        lookup = {"id": order_id}
        if not request.user.is_authenticated():
            lookup["key"] = request.session.session_key
        elif not request.user.is_staff:
            lookup["user_id"] = request.user.id
        return self.get(**lookup)
class ProductOptionManager(Manager):
    def as_fields(self):
        """
        Return a dict of product options as their field names and
        choices.
        """
        options = defaultdict(list)
        for option in self.all():
            options["option%s" % option.type].append(option.name)
        return options
class ProductVariationManager(Manager):
    use_for_related_fields = True
    def _empty_options_lookup(self, exclude=None):
        """
        Create a lookup dict of field__isnull for options fields.
        """
        if not exclude:
            exclude = {}
        return dict([("%s__isnull" % f.name, True)
            for f in self.model.option_fields() if f.name not in exclude])
    def create_from_options(self, options):
        """
        Create all unique variations from the selected options.
        """
        if options:
            options = OrderedDict(options)
            # Build all combinations of options.
            variations = [[]]
            for values_list in list(options.values()):
                variations = [x + [y] for x in variations for y in values_list]
            for variation in variations:
                # Lookup unspecified options as null to ensure a
                # unique filter.
                variation = dict(list(zip(list(options.keys()), variation)))
                lookup = dict(variation)
                lookup.update(self._empty_options_lookup(exclude=variation))
                try:
                    self.get(**lookup)
                except self.model.DoesNotExist:
                    self.create(**variation)
    def manage_empty(self):
        """
        Create an empty variation (no options) if none exist,
        otherwise if multiple variations exist ensure there is no
        redundant empty variation. Also ensure there is at least one
        default variation.
        """
        total_variations = self.count()
        if total_variations == 0:
            self.create()
        elif total_variations > 1:
            self.filter(**self._empty_options_lookup()).delete()
        try:
```

```
            self.get(default=True)
        except self.model.DoesNotExist:
            first_variation = self.all()[0]
            first_variation.default = True
            first_variation.save()
    def set_default_images(self, deleted_image_ids):
        """
        Assign the first image for the product to each variation that
        doesn't have an image. Also remove any images that have been
        deleted via the admin to avoid invalid image selections.
        """
        variations = self.all()
        if not variations:
            return
        image = variations[0].product.images.exclude(id__in=deleted_image_ids)
        if image:
            image = image[0]
        for variation in variations:
            save = False
            if str(variation.image_id) in deleted_image_ids:
                variation.image = None
                save = True
            if image and not variation.image:
                variation.image = image
                save = True
            if save:
                variation.save()
class ProductActionManager(Manager):
    use_for_related_fields = True
    def _action_for_field(self, field):
        """
        Increases the given field by datetime.today().toordinal()
        which provides a time scaling value we can order by to
        determine popularity over time.
        """
        timestamp = datetime.today().toordinal()
        action, created = self.get_or_create(timestamp=timestamp)
        setattr(action, field, getattr(action, field) + 1)
        action.save()
    def added_to_cart(self):
        """
        Increase total_cart when product is added to cart.
        """
        self._action_for_field("total_cart")
    def purchased(self):
        """
        Increase total_purchased when product is purchased.
        """
        self._action_for_field("total_purchase")
class DiscountCodeManager(Manager):
    def active(self, *args, **kwargs):
        """
        Items flagged as active and in valid date range if date(s) are
        specified.
        """
        n = now()
        valid_from = Q(valid_from__isnull=True) | Q(valid_from__lte=n)
        valid_to = Q(valid_to__isnull=True) | Q(valid_to__gte=n)
        valid = self.filter(valid_from, valid_to, active=True)
        return valid.exclude(uses_remaining=0)
    def get_valid(self, code, cart):
        """
        Items flagged as active and within date range as well checking
        that the given cart contains items that the code is valid for.
        """
        total_price_valid = (Q(min_purchase__isnull=True) |
```

```
                        Q(min_purchase__lte=cart.total_price()))
        discount = self.active().get(total_price_valid, code=code)
        products = discount.all_products()
        if products.count() > 0:
            if products.filter(variations__sku__in=cart.skus()).count() == 0:
                raise self.model.DoesNotExist
        return discount
```

```
"""
Checkout process.
"""
from __future__ import unicode_literals
from django.utils.translation import ugettext_lazy as _
from django.template.loader import get_template, TemplateDoesNotExist
from mezzanine.accounts import get_profile_for_user, ProfileNotConfigured
from mezzanine.conf import settings
from mezzanine.utils.email import send_mail_template
from cartridge.shop.models import Order
from cartridge.shop.utils import set_shipping, set_tax, sign
class CheckoutError(Exception):
    """
    Should be raised in billing/shipping and payment handlers for
    cases such as an invalid shipping address or an unsuccessful
    payment.
    """
    pass
def default_billship_handler(request, order_form):
    """
    Default billing/shipping handler - called when the first step in
    the checkout process with billing/shipping address fields is
    submitted. Implement your own and specify the path to import it
    from via the setting ``SHOP_HANDLER_BILLING_SHIPPING``.
    This function will typically contain any shipping calculation
    where the shipping amount can then be set using the function
    ``cartridge.shop.utils.set_shipping``. The Cart object is also
    accessible via ``request.cart``
    """
    if not request.session.get("free_shipping"):
        settings.use_editable()
        set_shipping(request, _("Flat rate shipping"),
                     settings.SHOP_DEFAULT_SHIPPING_VALUE)
def default_tax_handler(request, order_form):
    """
    Default tax handler - called immediately after the handler defined
    by ``SHOP_HANDLER_BILLING_SHIPPING``. Implement your own and
    specify the path to import it from via the setting
    ``SHOP_HANDLER_TAX``. This function will typically contain any tax
    calculation where the tax amount can then be set using the function
    ``cartridge.shop.utils.set_tax``. The Cart object is also
    accessible via ``request.cart``
    """
    settings.use_editable()
    set_tax(request, _("Tax"), 0)
def default_payment_handler(request, order_form, order):
    """
    Default payment handler - called when the final step of the
    checkout process with payment information is submitted. Implement
    your own and specify the path to import it from via the setting
    ``SHOP_HANDLER_PAYMENT``. This function will typically contain
    integration with a payment gateway. Raise
    cartridge.shop.checkout.CheckoutError("error message") if payment
    is unsuccessful.
    """
    pass
def default_order_handler(request, order_form, order):
    """
```

```
    Default order handler - called when the order is complete and
    contains its final data. Implement your own and specify the path
    to import it from via the setting ``SHOP_HANDLER_ORDER``.
    """
    pass
def initial_order_data(request, form_class=None):
    """
    Return the initial data for the order form, trying the following in
    order:
    - request.POST which is available when moving backward through the
      checkout steps
    - current order details in the session which are populated via each
      checkout step, to support user leaving the checkout entirely and
      returning
    - last order made by the user, via user ID or cookie
    - matching fields on an authenticated user and profile object
    """
    initial = {}
    if request.method == "POST":
        initial = dict(list(request.POST.items()))
        try:
            initial = form_class.preprocess(initial)
        except (AttributeError, TypeError):
            # form_class has no preprocess method, or isn't callable.
            pass
        # POST on first step won't include the "remember" checkbox if
        # it isn't checked, and it'll then get an actual value of False
        # when it's a hidden field - so we give it an empty value when
        # it's missing from the POST data, to persist it not checked.
        initial.setdefault("remember", "")
    # Look for order in current session.
    if not initial:
        initial = request.session.get("order", {})
    # Look for order in previous session.
    if not initial:
        lookup = {}
        if request.user.is_authenticated():
            lookup["user_id"] = request.user.id
        remembered = request.COOKIES.get("remember", "").split(":")
        if len(remembered) == 2 and remembered[0] == sign(remembered[1]):
            lookup["key"] = remembered[1]
        if lookup:
            previous = list(Order.objects.filter(**lookup).values())[:1]
            if len(previous) > 0:
                initial.update(previous[0])
    if not initial and request.user.is_authenticated():
        # No previous order data - try and get field values from the
        # logged in user. Check the profile model before the user model
        # if it's configured. If the order field name uses one of the
        # billing/shipping prefixes, also check for it without the
        # prefix. Finally if a matching attribute is callable, call it
        # for the field value, to support custom matches on the profile
        # model.
        user_models = [request.user]
        try:
            user_models.insert(0, get_profile_for_user(request.user))
        except ProfileNotConfigured:
            pass
        for order_field in form_class._meta.fields:
            check_fields = [order_field]
            for prefix in ("billing_detail_", "shipping_detail_"):
                if order_field.startswith(prefix):
                    check_fields.append(order_field.replace(prefix, "", 1))
            for user_model in user_models:
                for check_field in check_fields:
                    user_value = getattr(user_model, check_field, None)
                    if user_value:
```

```
                        if callable(user_value):
                            try:
                                user_value = user_value()
                            except TypeError:
                                continue
                        if not initial.get(order_field):
                            initial[order_field] = user_value
    # Set initial value for "same billing/shipping" based on
    # whether both sets of address fields are all equal.
    shipping = lambda f: "shipping_%s" % f[len("billing_"):]
    if any([f for f in form_class._meta.fields if f.startswith("billing_") and
        shipping(f) in form_class._meta.fields and
        initial.get(f, "") != initial.get(shipping(f), "")]):
        initial["same_billing_shipping"] = False
    # Never prepopulate discount code.
    try:
        del initial["discount_code"]
    except KeyError:
        pass
    return initial
def send_order_email(request, order):
    """
    Send order receipt email on successful order.
    """
    settings.use_editable()
    order_context = {"order": order, "request": request,
                     "order_items": order.items.all()}
    order_context.update(order.details_as_dict())
    try:
        get_template("shop/email/order_receipt.html")
    except TemplateDoesNotExist:
        receipt_template = "email/order_receipt"
    else:
        receipt_template = "shop/email/order_receipt"
        from warnings import warn
        warn("Shop email receipt templates have moved from "
             "templates/shop/email/ to templates/email/")
    send_mail_template(settings.SHOP_ORDER_EMAIL_SUBJECT,
                       receipt_template, settings.SHOP_ORDER_FROM_EMAIL,
                       order.billing_detail_email, context=order_context,
                       addr_bcc=settings.SHOP_ORDER_EMAIL_BCC or None)
# Set up some constants for identifying each checkout step.
CHECKOUT_STEPS = [{"template": "billing_shipping", "url": "details",
                   "title": _("Details")}]
CHECKOUT_STEP_FIRST = CHECKOUT_STEP_PAYMENT = CHECKOUT_STEP_LAST = 1
if settings.SHOP_CHECKOUT_STEPS_SPLIT:
    CHECKOUT_STEPS[0].update({"url": "billing-shipping",
                              "title": _("Address")})
    if settings.SHOP_PAYMENT_STEP_ENABLED:
        CHECKOUT_STEPS.append({"template": "payment", "url": "payment",
                               "title": _("Payment")})
        CHECKOUT_STEP_PAYMENT = CHECKOUT_STEP_LAST = 2
if settings.SHOP_CHECKOUT_STEPS_CONFIRMATION:
    CHECKOUT_STEPS.append({"template": "confirmation", "url": "confirmation",
                           "title": _("Confirmation")})
    CHECKOUT_STEP_LAST += 1
```

```
#МОДУЛЬ ПІДСИСТЕМИ ОПЛАТИ

from __future__
import
unicode_literals
                try:
                    from urllib.request import Request, urlopen
                    from urllib.error import URLError
                except ImportError:
```

```
    from urllib2 import Request, urlopen, URLError
import locale
from django.core.exceptions import ImproperlyConfigured
from django.http import QueryDict
from django.utils.http import urlencode
from mezzanine.conf import settings
from cartridge.shop.checkout import CheckoutError
PAYPAL_NVP_API_ENDPOINT_SANDBOX = 'https://api-3t.sandbox.paypal.com/nvp'
PAYPAL_NVP_API_ENDPOINT = 'https://api-3t.paypal.com/nvp'
try:
    PAYPAL_USER = settings.PAYPAL_USER
    PAYPAL_PASSWORD = settings.PAYPAL_PASSWORD
    PAYPAL_SIGNATURE = settings.PAYPAL_SIGNATURE
except AttributeError:
    raise ImproperlyConfigured("You need to define PAYPAL_USER, "
                               "PAYPAL_PASSWORD and PAYPAL_SIGNATURE "
                               "in your settings module to use the "
                               "paypal payment processor.")
def process(request, order_form, order):
    """
    Paypal direct payment processor.
    PayPal is picky.
    - https://cms.paypal.com/us/cgi-bin/?cmd=_render-content
      &content_ID=developer/e_howto_api_nvp_r_DoDirectPayment
    - https://cms.paypal.com/us/cgi-bin/?cmd=_render-content
      &content_ID=developer/e_howto_api_nvp_errorcodes
    Paypal requires the countrycode, and that it be specified in 2 single-
    byte characters. Import the COUNTRIES tuple-of-tuples, included below,
    and subclass OrderForm in my app, e.g.:
    from cartridge.shop.payment.paypal import COUNTRIES
    class MyOrderForm(OrderForm):
        def __init__(self, *args, **kwargs):
            super(OrderForm, self).__init__(*args, **kwrds)
            billing_country = forms.Select(choices=COUNTRIES)
            shipping_country = forms.Select(choices=COUNTRIES)
            self.fields['billing_detail_country'].widget = billing_country
            self.fields['shipping_detail_country'].widget = shipping_country
    Raise cartride.shop.checkout.CheckoutError("error message") if
    payment is unsuccessful.
    """
    trans = {}
    amount = order.total
    trans['amount'] = amount
    locale.setlocale(locale.LC_ALL, str(settings.SHOP_CURRENCY_LOCALE))
    currency = locale.localeconv()
    try:
        ipaddress = request.META['HTTP_X_FORWARDED_FOR']
    except:
        ipaddress = request.META['REMOTE_ADDR']
    if settings.DEBUG:
        trans['connection'] = PAYPAL_NVP_API_ENDPOINT_SANDBOX
    else:
        trans['connection'] = PAYPAL_NVP_API_ENDPOINT
    trans['configuration'] = {
        'USER': PAYPAL_USER,
        'PWD': PAYPAL_PASSWORD,
        'SIGNATURE': PAYPAL_SIGNATURE,
        'VERSION': '53.0',
        'METHOD': 'DoDirectPayment',
        'PAYMENTACTION': 'Sale',
        'RETURNFMFDETAILS': 0,
        'CURRENCYCODE': currency['int_curr_symbol'][0:3],
        'IPADDRESS': ipaddress,
    }
    data = order_form.cleaned_data
    trans['custBillData'] = {
```

```
        'FIRSTNAME': data['billing_detail_first_name'],
        'LASTNAME': data['billing_detail_last_name'],
        'STREET': data['billing_detail_street'],
        'CITY': data['billing_detail_city'],
        'STATE': data['billing_detail_state'],
        'ZIP': data['billing_detail_postcode'],
        'COUNTRYCODE': data['billing_detail_country'],
        # optional below
        'SHIPTOPHONENUM': data['billing_detail_phone'],
        'EMAIL': data['billing_detail_email'],
    }
    trans['custShipData'] = {
        'SHIPTONAME': (data['shipping_detail_first_name'] + ' ' +
                       data['shipping_detail_last_name']),
        'SHIPTOSTREET': data['shipping_detail_street'],
        'SHIPTOCITY': data['shipping_detail_city'],
        'SHIPTOSTATE': data['shipping_detail_state'],
        'SHIPTOZIP': data['shipping_detail_postcode'],
        'SHIPTOCOUNTRY': data['shipping_detail_country'],
    }
    trans['transactionData'] = {
        'CREDITCARDTYPE': data['card_type'].upper(),
        'ACCT': data['card_number'].replace(' ', ''),
        'EXPDATE': str(data['card_expiry_month'] + data['card_expiry_year']),
        'CVV2': data['card_ccv'],
        'AMT': trans['amount'],
        'INVNUM': str(order.id)
    }
    part1 = urlencode(trans['configuration']) + "&"
    part2 = "&" + urlencode(trans['custBillData'])
    part3 = "&" + urlencode(trans['custShipData'])
    trans['postString'] = (part1 + urlencode(trans['transactionData']) +
                           part2 + part3)
    trans['postString'] = trans['postString'].encode('utf-8')
    request_args = {"url": trans['connection'], "data": trans['postString']}
    # useful for debugging transactions
    # print trans['postString']
    try:
        all_results = urlopen(Request(**request_args)).read()
    except URLError:
        raise CheckoutError("Could not talk to PayPal payment gateway")
    parsed_results = QueryDict(all_results)
    state = parsed_results['ACK']
    if state not in ["Success", "SuccessWithWarning"]:
        raise CheckoutError(parsed_results['L_LONGMESSAGE0'])
    return parsed_results['TRANSACTIONID']
```

```
#МОДУЛЬ АВТОРИЗАЦІЇ
from __future__
import
unicode_literals
```

```
from future.builtins import str
try:
    from urllib.request import Request, urlopen
    from urllib.error import URLError
except ImportError:
    from urllib2 import Request, urlopen, URLError
from django.core.exceptions import ImproperlyConfigured
from django.utils.http import urlencode
from mezzanine.conf import settings
from cartridge.shop.checkout import CheckoutError
AUTH_NET_LIVE = 'https://secure.authorize.net/gateway/transact.dll'
AUTH_NET_TEST = 'https://test.authorize.net/gateway/transact.dll'
try:
    AUTH_NET_LOGIN = settings.AUTH_NET_LOGIN
    AUTH_NET_TRANS_KEY = settings.AUTH_NET_TRANS_KEY
except AttributeError:
    raise ImproperlyConfigured("You need to define AUTH_NET_LOGIN and "
```

```
                                "AUTH_NET_TRANS_KEY in your settings module "
                                "to use the authorizenet payment processor.")
def process(request, order_form, order):
    """
    Raise cartridge.shop.checkout.CheckoutError("error message") if
    payment is unsuccessful.
    """
    trans = {}
    amount = order.total
    trans['amount'] = amount
    if settings.DEBUG:
        trans['connection'] = AUTH_NET_TEST
    else:
        trans['connection'] = AUTH_NET_LIVE
    trans['authorize_only'] = False
    trans['configuration'] = {
        'x_login': AUTH_NET_LOGIN,
        'x_tran_key': AUTH_NET_TRANS_KEY,
        'x_version': '3.1',
        'x_relay_response': 'FALSE',
        'x_test_request': 'FALSE',
        'x_delim_data': 'TRUE',
        'x_delim_char': '|',
        # could be set to AUTH_ONLY to only authorize but not capture payment
        'x_type': 'AUTH_CAPTURE',
        'x_method': 'CC',
    }
    data = order_form.cleaned_data
    trans['custBillData'] = {
        'x_first_name': data['billing_detail_first_name'],
        'x_last_name': data['billing_detail_last_name'],
        'x_address': data['billing_detail_street'],
        'x_city': data['billing_detail_city'],
        'x_state': data['billing_detail_state'],
        'x_zip': data['billing_detail_postcode'],
        'x_country': data['billing_detail_country'],
        'x_phone': data['billing_detail_phone'],
        'x_email': data['billing_detail_email'],
    }
    trans['custShipData'] = {
        'x_ship_to_first_name': data['shipping_detail_first_name'],
        'x_ship_to_last_name': data['shipping_detail_last_name'],
        'x_ship_to_address': data['shipping_detail_street'],
        'x_ship_to_city': data['shipping_detail_city'],
        'x_ship_to_state': data['shipping_detail_state'],
        'x_ship_to_zip': data['shipping_detail_postcode'],
        'x_ship_to_country': data['shipping_detail_country'],
    }
    trans['transactionData'] = {
        'x_amount': amount,
        'x_card_num': data['card_number'],
        'x_exp_date': '{month}/{year}'.format(month=data['card_expiry_month'],
                                              year=data['card_expiry_year']),
        'x_card_code': data['card_ccv'],
        'x_invoice_num': str(order.id)
    }
    part1 = urlencode(trans['configuration']) + "&"
    part2 = "&" + urlencode(trans['custBillData'])
    part3 = "&" + urlencode(trans['custShipData'])
    trans['postString'] = (part1 + urlencode(trans['transactionData']) +
                           part2 + part3)
    request_args = {"url": trans['connection'],
                    "data": trans['postString'].encode('utf-8')}
    try:
        all_results = urlopen(Request(**request_args)).read()
    except URLError:
        raise CheckoutError("Could not talk to authorize.net payment gateway")
```

```
    parsed_results = all_results.decode('utf-8').split(
        trans['configuration']['x_delim_char'])
    # response and response_reason_codes with their meaning here:
    # http://www.authorize.net/support/merchant/Transaction_Response/
    # Response_Reason_Codes_and_Response_Reason_Text.htm
    # not exactly sure what the reason code is
    response_code = parsed_results[0]
    # reason_code = parsed_results[1]
    # response_reason_code = parsed_results[2]
    # response_text = parsed_results[3]
    # transaction_id = parsed_results[6]
    success = response_code == '1'
    if not success:
        raise CheckoutError("Transaction declined: " + parsed_results[2])
    return parsed_results[6]
```

МОДУЛЬ АДМІНКИ

```
from copy import
deepcopy

from django.contrib import admin
from django.contrib.admin.templatetags.admin_static import
static
from django.db.models import ImageField
from django.utils.translation import ugettext_lazy as _
from mezzanine.conf import settings
from mezzanine.core.admin import (DisplayableAdmin,

TabularDynamicInlineAdmin,

BaseTranslationModelAdmin)
from mezzanine.pages.admin import PageAdmin
from cartridge.shop.fields import MoneyField
from cartridge.shop.forms import ProductAdminForm,
ProductVariationAdminForm
from cartridge.shop.forms import
ProductVariationAdminFormset
from cartridge.shop.forms import DiscountAdminForm,
ImageWidget, MoneyWidget
from cartridge.shop.models import Category, Product,
ProductImage
from cartridge.shop.models import ProductVariation,
ProductOption, Order
from cartridge.shop.models import OrderItem, Sale,
DiscountCode
from cartridge.shop.views import HAS_PDF
# Lists of field names.
option_fields = [f.name for f in
ProductVariation.option_fields()]
_flds = lambda s: [f.name for f in Order._meta.fields if
f.name.startswith(s)]
billing_fields = _flds("billing_detail")
shipping_fields = _flds("shipping_detail")
#################
#  CATEGORIES  #
#################
# Categories fieldsets are extended from Page fieldsets,
since
# categories are a Mezzanine Page type.
category_fieldsets = deepcopy(PageAdmin.fieldsets)
category_fieldsets[0][1]["fields"][3:3] = ["content",
"products"]
category_fieldsets += ((_("Product filters"), {
    "fields": ("sale", ("price_min", "price_max"),
"combined"),
    "classes": ("collapse-closed",)},),)
if settings.SHOP_CATEGORY_USE_FEATURED_IMAGE:
    category_fieldsets[0][1]["fields"].insert(3,
"featured_image")
```

```
# Options are only used when variations are in use, so only
provide
# them as filters for dynamic categories when this is the
case.
if settings.SHOP_USE_VARIATIONS:
    category_fieldsets[-1][1]["fields"] = (("options",) +

category_fieldsets[-1][1]["fields"])
class CategoryAdmin(PageAdmin):
    fieldsets = category_fieldsets
    formfield_overrides = {ImageField: {"widget":
ImageWidget}}
    filter_horizontal = ("options", "products",)
################
#  VARIATIONS  #
################
# If variations aren't used, the variation inline should
always
# provide a single inline for managing the single variation
per
# product.
variation_fields = ["sku", "num_in_stock", "unit_price",
                    "sale_price", "sale_from", "sale_to",
"image"]
if settings.SHOP_USE_VARIATIONS:
    variation_fields.insert(1, "default")
    variations_max_num = None
    variations_extra = 0
else:
    variations_max_num = 1
    variations_extra = 1
class ProductVariationAdmin(admin.TabularInline):
    verbose_name_plural = _("Current variations")
    model = ProductVariation
    fields = variation_fields
    max_num = variations_max_num
    extra = variations_extra
    formfield_overrides = {MoneyField: {"widget":
MoneyWidget}}
    form = ProductVariationAdminForm
    formset = ProductVariationAdminFormset
    ordering = ["option%s" % i for i in
settings.SHOP_OPTION_ADMIN_ORDER]
class ProductImageAdmin(TabularDynamicInlineAdmin):
    model = ProductImage
    formfield_overrides = {ImageField: {"widget":
ImageWidget}}
###############
#  PRODUCTS  #
###############
product_fieldsets = deepcopy(DisplayableAdmin.fieldsets)
product_fieldsets[0][1]["fields"].insert(2, "available")
product_fieldsets[0][1]["fields"].extend(["content",
"categories"])
product_fieldsets = list(product_fieldsets)
other_product_fields = []
if settings.SHOP_USE_RELATED_PRODUCTS:
    other_product_fields.append("related_products")
if settings.SHOP_USE_UPSELL_PRODUCTS:
    other_product_fields.append("upsell_products")
if len(other_product_fields) > 0:
    product_fieldsets.append((_("Other products"), {
        "classes": ("collapse-closed",),
        "fields": tuple(other_product_fields)}))
product_list_display = ["admin_thumb", "title", "status",
"available",
                        "admin_link"]
```

```
product_list_editable = ["status", "available"]
# If variations are used, set up the product option fields
for managing
# variations. If not, expose the denormalised price fields
for a product
# in the change list view.
if settings.SHOP_USE_VARIATIONS:
    product_fieldsets.insert(1, (_("Create new
variations"),
        {"classes": ("create-variations",), "fields":
option_fields}))
else:
    extra_list_fields = ["sku", "unit_price", "sale_price",
"num_in_stock"]
    product_list_display[4:4] = extra_list_fields
    product_list_editable.extend(extra_list_fields)
class ProductAdmin(DisplayableAdmin):
    class Media:
        js =
(static("cartridge/js/admin/product_variations.js"),)
        css = {"all":
(static("cartridge/css/admin/product.css"),)}
    list_display = product_list_display
    list_display_links = ("admin_thumb", "title")
    list_editable = product_list_editable
    list_filter = ("status", "available", "categories")
    filter_horizontal = ("categories",) +
tuple(other_product_fields)
    search_fields = ("title", "content",
"categories__title",
                     "variations__sku")
    inlines = (ProductImageAdmin, ProductVariationAdmin)
    form = ProductAdminForm
    fieldsets = product_fieldsets
    def save_model(self, request, obj, form, change):
        """
        Store the product object for creating variations in
save_formset.
        """
        super(ProductAdmin, self).save_model(request, obj,
form, change)
        self._product = obj
    def save_formset(self, request, form, formset, change):
        """
        Here be dragons. We want to perform these steps
sequentially:
        - Save variations formset
        - Run the required variation manager methods:
          (create_from_options, manage_empty, etc)
        - Save the images formset
        The variations formset needs to be saved first for
the manager
        methods to have access to the correct variations.
The images
        formset needs to be run last, because if images are
deleted
        that are selected for variations, the variations
formset will
        raise errors when saving due to invalid image
selections. This
        gets addressed in the set_default_images method.
        An additional problem is the actual ordering of the
inlines,
        which are in the reverse order for achieving the
above. To
        address this, we store the images formset as an
attribute, and
```

```
        then call save on it after the other required steps
have
        occurred.
        """
        # Store the images formset for later saving,
otherwise save the
        # formset.
        if formset.model == ProductImage:
            self._images_formset = formset
        else:
            super(ProductAdmin, self).save_formset(request,
form, formset,
                                                   change)
        # Run each of the variation manager methods if
we're saving
        # the variations formset.
        if formset.model == ProductVariation:
            # Build up selected options for new variations.
            options = dict([(f, request.POST.getlist(f))
for f in option_fields
                            if request.POST.getlist(f)])
            # Create a list of image IDs that have been
marked to delete.
            deleted_images = [request.POST.get(f.replace("-
DELETE", "-id"))
                for f in request.POST
                if f.startswith("images-") and
f.endswith("-DELETE")]
            # Create new variations for selected options.

self._product.variations.create_from_options(options)
            # Create a default variation if there are none.
            self._product.variations.manage_empty()
            # Remove any images deleted just now from
variations they're
            # assigned to, and set an image for any
variations without one.

self._product.variations.set_default_images(deleted_images)
            # Save the images formset stored previously.
            super(ProductAdmin, self).save_formset(request,
form,

self._images_formset, change)
            # Run again to allow for no images existing
previously, with
            # new images added which can be used as
defaults for variations.

self._product.variations.set_default_images(deleted_images)
            # Copy duplicate fields (``Priced`` fields)
from the default
            # variation to the product.
            self._product.copy_default_variation()
            # Save every translated fields from
``ProductOption`` into
            # the required ``ProductVariation``
            if settings.USE_MODELTRANSLATION:
                from collections import OrderedDict
                from modeltranslation.utils import
(build_localized_fieldname
                                                      as
_loc)
                for opt_name in options:
                    for opt_value in options[opt_name]:
                        opt_obj =
ProductOption.objects.get(type=opt_name[6:],
```

```
name=opt_value)
                        params = {opt_name: opt_value}
                        for var in
self._product.variations.filter(**params):
                            for code in
OrderedDict(settings.LANGUAGES):
                                setattr(var, _loc(opt_name,
code),
                                        getattr(opt_obj,
_loc('name', code)))
                            var.save()
class ProductOptionAdmin(BaseTranslationModelAdmin):
    ordering = ("type", "name")
    list_display = ("type", "name")
    list_display_links = ("type",)
    list_editable = ("name",)
    list_filter = ("type",)
    search_fields = ("type", "name")
    radio_fields = {"type": admin.HORIZONTAL}
class OrderItemInline(admin.TabularInline):
    verbose_name_plural = _("Items")
    model = OrderItem
    extra = 0
    formfield_overrides = {MoneyField: {"widget":
MoneyWidget}}
def address_pairs(fields):
    """
    Zips address fields into pairs, appending the last
field if the
    total is an odd number.
    """
    pairs = list(zip(fields[::2], fields[1::2]))
    if len(fields) % 2:
        pairs.append(fields[-1])
    return pairs
order_list_display = ("id", "billing_name", "total",
"time", "status",
                      "transaction_id")
if HAS_PDF:
    order_list_display += ("invoice",)
class OrderAdmin(admin.ModelAdmin):
    class Media:
        css = {"all":
(static("cartridge/css/admin/order.css"),)}
    ordering = ("status", "-id")
    list_display = order_list_display
    list_editable = ("status",)
    list_filter = ("status", "time")
    list_display_links = ("id", "billing_name",)
    search_fields = (["id", "status", "transaction_id"] +
                     billing_fields + shipping_fields)
    date_hierarchy = "time"
    radio_fields = {"status": admin.HORIZONTAL}
    inlines = (OrderItemInline,)
    formfield_overrides = {MoneyField: {"widget":
MoneyWidget}}
    fieldsets = (
        (_("Billing details"), {"fields":
address_pairs(billing_fields)}),
         (_("Shipping details"), {"fields":
address_pairs(shipping_fields)}),
        (None, {"fields": ("additional_instructions",
("shipping_total",
            "shipping_type"), ('tax_total', 'tax_type'),
             ("discount_total", "discount_code"),
"item_total",
```

```
            ("total", "status"), "transaction_id")}),
    )
    def change_view(self, *args, **kwargs):
        if kwargs.get("extra_context", None) is None:
            kwargs["extra_context"] = {}
        kwargs["extra_context"]["has_pdf"] = HAS_PDF
        return super(OrderAdmin, self).change_view(*args,
**kwargs)
class SaleAdmin(admin.ModelAdmin):
    list_display = ("title", "active", "discount_deduct",
"discount_percent",
        "discount_exact", "valid_from", "valid_to")
    list_editable = ("active", "discount_deduct",
"discount_percent",
        "discount_exact", "valid_from", "valid_to")
    filter_horizontal = ("categories", "products")
    formfield_overrides = {MoneyField: {"widget":
MoneyWidget}}
    form = DiscountAdminForm
    fieldsets = (
        (None, {"fields": ("title", "active")}),
        (_("Apply to product and/or products in
categories"),
            {"fields": ("products", "categories")}),
        (_("Reduce unit price by"),
            {"fields": (("discount_deduct",
"discount_percent",
            "discount_exact"),)}),
        (_("Sale period"), {"fields": (("valid_from",
"valid_to"),)}),
    )
class DiscountCodeAdmin(admin.ModelAdmin):
    list_display = ("title", "active", "code",
"discount_deduct",
        "discount_percent", "min_purchase",
"free_shipping", "valid_from",
        "valid_to")
    list_editable = ("active", "code", "discount_deduct",
"discount_percent",
        "min_purchase", "free_shipping", "valid_from",
"valid_to")
    filter_horizontal = ("categories", "products")
    formfield_overrides = {MoneyField: {"widget":
MoneyWidget}}
    form = DiscountAdminForm
    fieldsets = (
        (None, {"fields": ("title", "active", "code")}),
        (_("Apply to product and/or products in
categories"),
            {"fields": ("products", "categories")}),
        (_("Reduce unit price by"),
            {"fields": (("discount_deduct",
"discount_percent"),)}),
        (None, {"fields": (("min_purchase",
"free_shipping"),)}),
        (_("Valid for"),
            {"fields": (("valid_from", "valid_to",
"uses_remaining"),)}),
    )
admin.site.register(Category, CategoryAdmin)
admin.site.register(Product, ProductAdmin)
if settings.SHOP_USE_VARIATIONS:
    admin.site.register(ProductOption, ProductOptionAdmin)
admin.site.register(Order, OrderAdmin)
admin.site.register(Sale, SaleAdmin)
admin.site.register(DiscountCode, DiscountCodeAdmin)
```