Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Проектування освітньої системи «STUDUCK»»

Студента 2 курсу, 6м групи, спеціальності 121 «Інженерія програмного забезпечення», спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Струка Владислава Сергійовича

Науковий керівник кандидат технічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Рзаєва Світлана Леонідівна

Гарант освітньої програми доктор технічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Криворучко Олена Володимирівна

КИЇВ – 2020

ЗМІСТ

					КНТЕУ 121-06-18.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В		19.10.20	*Проектування освітньої системи «STUDUCK»*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Рзаєва С.Л.		19.10.20		*Зміст*	2	49
Гарант		Криворучко О.В		19.10.20		*Факультет інформаційних технологій 2м курс, 6 група*		
Розробив		Струк В.С.		19.10.20	*Зміст*			

ВСТУП

У сучасному суспільстві визначальним стовпом формування людини є її освіта. Від рівня освіти людини напряму залежить її майбутнє: місце роботи, місце проживання, положення у суспільстві та навіть яка буде його друга половинка і багато іншого. Проте освіта це не лише отриманий диплом чи довідка про закінчення курсу, це перш за все отримані людиною знання та навички. Одним із найцінніших місць для отримання людиною знань є університет в якому навчають багато чому за досить малий проміжок часу, що часом може бути великою проблемою для студента так як далеко не кожен спроможен продуктивно засвоювати великі об’єми інформації різного типу. Для успішного засвоєння великого потоку інформації життєво необхідне її періодичне систематизування. В наш, стрімкий на розвиток, час кожен день відбуваються якісь відкриття чи переосмислення вже наявних знань, тому при систематизації подібних даних варто дотримуватись певних правил:

- позбавлення від всього непотрібного (не важливого);
- актуалізація даних що вже використовуються;
- поділ великих об’ємів даних на менші, розділені між собою;
- використання певних правил, що залежать від виду інформації яка систематизується.

Систематизація є невід’ємною частиною нашого життя, так як нам пощастило жити у час Інтернету, який дає змогу отримати будь-що майже будь-де. Об’єми даних, що існують в інтернеті, цілком можна

вважати безмежними так як майже кожен має можливість додавати свої данні, цим багато хто і користується напряму. Хоча інформацію в інтернеті і можна теоретично вважати структурованою в глобальному плані, проте в локальній вибірці вся інформація хаотично розкидана по всій всесвітній мережі. Її систематизувати практично неможливо, проте можливо створювати спеціалізовані форуми, портали, сайти та сервіси в яких всі необхідні данні вже будуть структуровані за певними правилами.

Одним із найкращих прикладів систематизування інформації можна привести інформаційні або ж освітні системи, які прийняли на сьогоднішній день як правило вигляд мобільних додатків або веб-сервісів. Зараз за допомогою подібних сервісів можна вивчити майже все, від англійської мови до ядерної фізики, у зручній для користувача формі будь це відеоматеріал статті, презентації чи навіть у формі звукозапису. Подібний підхід значно спрощує поріг входження для користувача так як він швидше засвоює інформацію у вподобаному йому вигляді та може відсіяти ту, що не потребує у даний момент. Також концентрація усієї необхідної у межах курсу інформації в одному місці значно спрощує пошук пов’язаних між собою даних. Здавалось би що це і є той самий університет у смартфоні чи браузері, проте відсутність подібних масштабних рішень для університетів навіть дивує. Саме тому автором було вирішено розробити мобільну освітньо-інформаційну систему націлену на використання студентами Київського національного торговельно-економічного університету, яка могла б стати невід’ємним асистентом для кожного, протягом студентського життя.

Актуальність: важливість освіти у сучасному світі важко переоцінити, при цьому комплексних програмних рішень у вигляді університетських освітніх систем під час пошуку аналогів виявлено не

було, не дивлячись на велику кількість як університетів, так і студентів, що в них навчаються.

Метою дослідження є проведення дослідження освітніх систем, аналіз їх архітектури та функціоналу з подальшою розробкою власної освітньої системи під назвою «STUDUCK» у програмному інструментарії для розробки «Solar2D» із використанням мови програмування Lua.

Об'єкт дослідження: освітній процес.

Предмет дослідження: розробка мобільного додатку у "Solar2D", з використанням мови програмування Lua.

Задачі дослідження:

- дослідити актуальність використання сучасних мобільних технологій у сфері освіти;
- проаналізувати можливі інструменти для розробки мобільних додатків та мови програмування що в них використовуються;
- сформувати технічне завдання на розробку освітньої системи «STUDUCK»;
- розробити архітектуру додатку освітньої системи «STUDUCK»;
- спроектувати користувацький інтерфейс для додатку освітньої системи «STUDUCK».

Наукова новизна дослідження полягає в розробці комплексного рішення у створенні освітньої системи аналогів якої під час пошуку не було знайдено.

Методи дослідження: порівняння; модулювання.

РОЗДІЛ 1.
ВІДОМОСТІ ЩОДО ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ ОСВІТНЬОЇ СИСТЕМИ

1.1 Освітні системи у мобільному середовищі

Людина усе своє життя проводить у навчанні. З народження вона вчиться ходити та розмовляти потім її ведуть у дитячий садочок, де її вчать базовим речам. З настанням 6-7 років дитина проводить наступні дев’ять або дванадцять років у школі навчаючись важливим предметам загального плану, які є необхідними кожній людині для існування у сучасному суспільстві. Після цього іде такий важливий етап як вступ до вищого навчального закладу, де вже кожного навчають цілком конкретним речам в залежності від обраної спеціальності, так як саме обрана спеціальність та успішність студента у навчанні визначить його майбутній соціальний статус у суспільстві навчаючись надалі вже лише необхідним речам в залежності від ситуації. Хоча кожний аспект є важливим, у формуванні особистості, проте саме навчання у вищому навчальному закладі дає змогу набути професію, налаштувати соціальні зв’язки та підготувати до справжнього дорослого життя.

Поки людина усе життя навчається, з недавнього часу активну участь у цьому приймають сучасні технології, в основному у вигляді смарфонів. Більш-менш сучасний смартфон є майже у кожної людини у розвинутій та не дуже країні. За даними які представила компанія Google на конференції Google I/O на 2019 рік у всьому світі більше ніж 2,5 мільярди активних пристроїв на операційній системі Android. У вересні 2015 року цей показник складав 1,4 мільярди. Їх прямі

конкуренти Apple заявляють про 1,4 мільярди активних мобільних пристроїв серед яких iPhone складають 900 мільйонів. Маючі такі дані можна дійти висновку, що на даний момент у світі є майже 3 мільярди активних мобільних пристроїв враховуючи лише дві основні операційні системи та не враховуючи ультрамобільні нетбуки та мобільні пристрої на операційних системах Windows Phone та інших. При цьому с кожним днем різноманітність пристроїв лише збільшується, що покриває все більшу і більшу частину цільового ринку захоплюючи світ та приходячи на заміну стандартним настільним рішенням.

У зв'язку з таким активним поширенням та використанням мобільних технологій, не є дивним, що окрім розважальних додатків у вигляді ігор на платформу також часто почали розроблювати і різноманітні програмні інструменти, як спеціальні, розроблені підприємством, для окремих своїх співробітників, так і загального призначення. Із великого різноманіття додатків на відповідних магазинах додатків нерідко можна зустріти різноманітні навчальні додатки, мета яких надати зручний інструментарій для ведення навчального процесу з метою отримання користувачем знань часто із певних, вузьких спеціальностей. Подібний підхід є доволі популярним у користувачів, так як вони майже завжди мають при собі свій смартфон для доступу у додаток, та можуть його використовувати у зручний для момент чи то під час тривалої поїздки, чи при звичайному очікувані де-небудь при цьому не маючи необхідності їхати до навчальних закладів. При такій зручній формі мобільного додатку цілком раціонально постає питання відсутності подібних рішень для вищих навчальних закладів, адже було б доволі зручно для студентів із «державою в смартфоні» також мати кишенькову версію «університету в смартфоні». Для подібного проекту реалізації

лише навчального додатку буде недостатньо, саме тому подібний проект буде класифікуватись як освітня система.

Освітня система зазвичай розглядається як об’єднання в цілісну систему незалежних програм та інститутів, що забезпечують як формальну, так і неформальну освіту або ж як сукупність навчальних закладів, науково-виробничих підприємств, державних органів управління освітою та самоврядування у галузі освіти, проте в рамках цього проекту освітня система буде розглядатись лише як програмне забезпечення. Таким чином, під освітньою системою розуміється узгоджена сукупність навчальних матеріалів, засобів їх розробки, зберігання, передачі і доступу до них, призначена для цілей навчання і заснована на використанні інформаційними технологіями. За своїм масштабом освітні системи у сфері вищої освіти може бути реалізовані у масштабах кафедри, університету або ж напряму підготовки. До основних функцій освітньої системи можна винести:

- доступ до навчальних матеріалів;
- самотестування і контроль знань;
- пошук інформації;
- конференцзв’язок;

Із подібного функціоналу можна зробити висновок, що освітня система це в більшій суті суміш інформаційної системи та навчального додатку із додатковим функціоналом.

Існує велике різноманіття додатків на тему освіти, проте майже всі вони являються навчальними додатками, де просто надається матеріал для вивчення, у кращому випадку є перевірка отриманих знань. Із таких прикладів можна виділити «Khan Academy» та «Лекторий МФТИ». Виділити їх можна завдяки різноманітним курсам (рис. 1.1.) коли

більшість інших навчальних додатків націлені на одну окрему спеціальність. На відміну від навчальних додатків «Школьник» є чудовим прикладом освітньої системи. Досягається це завдяки функціоналу, у додатку є можливість переглянути розклад, останні новини, зв’язатися із іншими учасниками освітнього процесу та отримати доступ до навчальних матеріалів. Як і усе у світі, попри усі переваги «Школьник» має недоліки, один із найголовніших це його відсутність у маркеті, що не уможливлює легальне використання додатку.

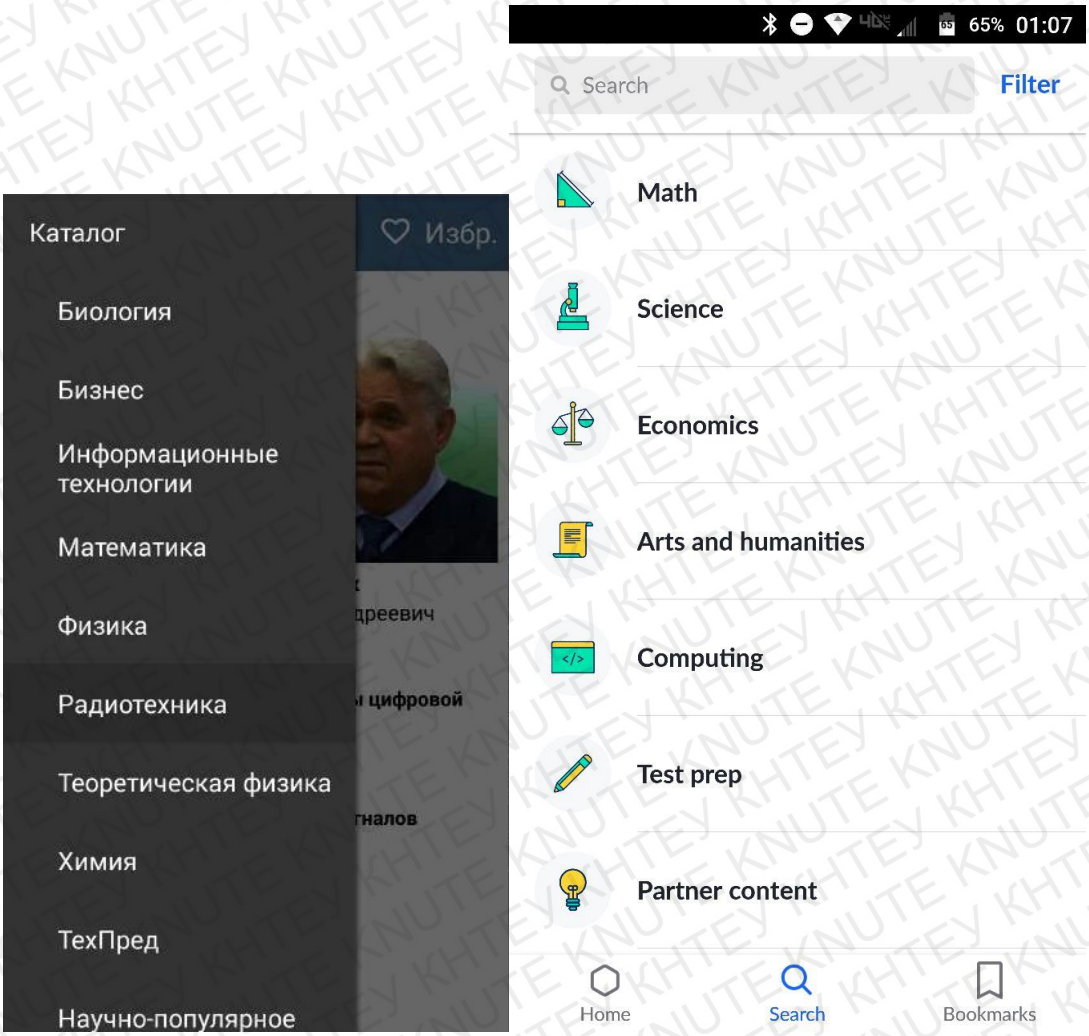


Рис. 1.1. Приклад різноманіття курсів у додатках «Лекторий МФТИ» та «Khan Academy»

1.2 Функціональні характеристики мови Lua

З самого створення перших машин що піддавалися програмуванню людина була вимушена адаптуватись сама, або ж адаптувати свої інструменти. Так як що тогочасні ЕОМ що сьогоднішні комп'ютери розуміють лише нулі та одиниці з булевої алгебри, людина була змушена або сама почати думати як машина, або ж розробити такий інструмент, що дав би змогу перетворювати зрозумілу мову для людини на машинний код. З цією метою був винайдений компілятор або як тоді його називали програмуюча програма так як тоді програмою вважався лише машинний код. Проте перетворювати звичайну розмовну мову досить складно якщо зовсім неможливо у зв'язку з варіативністю її використання та умовностям, тож для компіляторів існували свої мови – мови програмування.

Хоча історія комп'ютерної техніки і вважається відносно короткою, проте за цей час було вигадано більше ніж вісім тисяч різноманітних мов програмування і їх кількість з кожним роком лише росте. Їх згруповують по різним категоріям:

- за парадигмами;
- за окремою ознакою;
- неповнофункціональні;
- езотермічні.

В свою чергу такі основні категорії як за парадигмами та за окремою ознакою поділяються на додаткові підгрупи. За парадигмами можна поділити на:

- Аспектно-орієновані (AspectC++, AspectJ, AspectLua).
- Структурні (Basic, Pascal, Fortran).
- Процедурні (PHP, Lua, C, Golang).

- Логічні (Prolog, Mercury).
- Об’єктно-орієнтовані (C++, Haxe, Java, Python, C#).
- Функціональні (F#, Lisp, Erlang, Haskell).
- Мультипарадигмові (C++, Kotlin, Python, Swift).

В той же час за окремими ознаками поділяються на:

- Графічні мови (Visual dataFlex.)
- Мови для промислової автоматизації (FDB, IL, LadderDiagram).
- Стекові(Forth, Joy, Cat, PostScript).
- Паралельні (Clojure, Curry, Limbo).

І це лише мала частина від усього різноманіття мов серед яких своє скромне місце займає Lua. На початок її розробки у 1993 році Lua планувалась як домашня мова для двох доволі специфічних проектів, натомість зараз вона широко використовується у всіх областях які можуть отримати перевагу від використання простої, розширюваної, портативної та ефективної скриптової мови, такі як вбудовані системи, мобільні пристрої та, звісно ж, ігри.

З самого початку Lua розроблювалась як мова для інтеграції з програмним забезпеченням, що написане на C/C++ та інших загальноприйнятих мовах. Ця інтеграція дає багато переваг. Lua – це крихітна та проста мова, частково через те, що вона не намагається перевершити Су тому, у чому він вже досить хорош, наприклад: височайша швидкодія, низькорівневі операції та взаємодія зі стороннім програмним забезпеченням. Для цих задач Lua покладається на C. Lua в свою чергу пропонує як раз те, для чого C не досить добре підходить: висока ступінь незалежності від апаратного забезпечення, динамічні структури, відсутність збитковості і легкого тестування та налагодження. Для таких цілей Lua має у своєму розпорядженні безпечне оточення,

автоматичне управління пам’яттю та зручними засобами для роботи із рядками та іншими видами даних з розміром що динамічно змінюється.

Частина сили Lua іде від її бібліотек, і це не випадково, бо саме розширюваність є однією із самих сильних її сторін. Багато якостей мови вносять у це свій вклад. Динамічна типізація забезпечує високу ступінь поліморфізму. Автоматичне управління пам’яттю спрощує великий спектр у процесі розробки. Функції вищого порядку та анонімні функції забезпечують високу ступінь параметризації, роблячи функції більш універсальними.

Звісно, Lua – не єдина скриптова мова. Існують і багато інших мов, що можуть бути використані для тих же цілей, проте Lua надає набір можливостей, які роблять її кращим вибором для багатьох задач і які придають їй свій унікальний профіль:

- Розширюваність;
- Простота;
- Ефективність;
- Портативність;

Як результат Lua є чудовим вибором для використання і у якості додаткової мови, і у якості основної для реалізації багатьох можливих сценаріїв на самих різноманітних платформах.

1.3 Solar2D

Паралельно із розвитком мов програмування розвивались і засоби для ведення розробки програмного забезпечення. Кожної миті покращувались як вже існуючі інструменти так і створювались нові. Різноманітні текстові редактори, тяжкі та потужні інтегровані середовища розробки, більш вузько направлені SDK та цілі окремі двигуни які подекуди можуть уміщати у собі всі інші інструменти.

Для різних задач використовуються різні інструменти. Для великих ігор це спеціальні ігрові двигуни, для великих додатків це інтегровані середовища розробки для кожної окремої платформи в свою чергу, для реалізацій не самих амбіціозних проектів зручно використовувати SDK так як використання подібного інструменту значно прискорює прототипування відносно використанню інтегрованих середовищ розробки та часто має інші різноманітні переваги.

Так як розроблюваний додаток націлений на використання у мобільних системах, тож і інструмент для розробки повинен бути націлений на розробку для мобільних систем. Для кожної платформи можна використовувати окремі середовища розробки, проте такий підхід значно збільшить час розробки на кожну окрему платформу тож задля швидшого охоплення різних операційних систем доцільно буде використовувати кросс-платформний інструмент. Їх, звісно, на даний момент існує вже досить багато, і з кожним днем їх кількість лише збільшується, що дає змогу обрати той інструмент, що максимально підходитиме для реалізації конкретних задач.

Для написання додатків на мобільні операційні системи існує достатньо різноманітних кросс-платформних інструментів наприклад Haxe.IO, GodotEngine, LÖVE, Solar2D, Appgamekit. Проте всі вони мають як свої переваги так і недоліки. Appgamekitдосить потужний інструмент проте він платний і при цьому його підтримка на доволі низькому рівні, а використання BASICподібної мови значно ускладнює та сповільнює процес розробки. GodotEngine в свою чергу є безкоштовним та швидко розвивається проте він націлений на розробку перш за все ігор, а через швидкий темп розвитку існує велика кількість технічних недоліків які генерують велику кількість багів. Серед таких потужних варіантів

Solar2D та LÖVE виділяються зі своїм обмеженням розробки лише 2д додатків. Проте вони використовують легку у вивченні мову програмування Lua та вже досить довго знаходяться на ринку щоб працювати стабільно. Їх можна було б вважати цілком однаковими, проте їх розрізняє один нюанс: LÖVE націлений на розробку для Windows з підтримкою мобільних платформ, а Solar2Dнацілений на розробку для мобільних платформ з підтримкою Windows. Таким чином вибір можна вважати очевидним.

Якщо коротко характеризувати то Solar2D– це ігровий двигун на основі Lua, що орієнтований на простоту впровадження та використання. Він являється проектом з повністю відкритим початковим кодом, створений на основі добре зарекомендувавши себе та широко використовуваного ігрового двигуна CoronaSDK, котрий більше не має комерційної підтримки. Із переваг цього інструменту відносяться:

- крос-платформність – швидка реалізація підтримки iOS, tvOS, Android, AndroidTV, macOS, Windows, Linux, HTML5;
- використання мови програмування Lua з її перевагами;
- гнучка підтримка плагінів – що розширюють функціональність;
- підтримка нативних бібліотек – що дає більший доступ до різноманітних можливостей системи;
- стабільність системи – завдяки налагодженому за 10 років розробки процесу;
- повна безкоштовність – відсутність будь-яких прихованих комісій та зборів незалежно від команди розробки;
- відсутність анонімного збору даних – що спрощує публікацію продукту;
- активна спільнота розробників;

- OpenSource – що дає змогу детального налаштування інструментарію на рівні ядра;
- тестування на льоту – що дозволяє проводити швидке тестування на мобільних платформах завдяки використанню Corona Live Server.

Подібний підхід до реалізації інструментарію позитивно впливає на популярність самого двигуна, що цілком заслужено робить Solar2Dодним із кращих варіантів при виборі легкого у засвоєнні крос-платформного інструменту для розробки програмного забезпечення.

1.4 Висновок до розділу 1

Дослідивши та проаналізувавши сучасні тенденції розвитку мобільних пристроїв був проведений аналіз та порівняння вже існуючих аналогів програмного забезпечення. Розглянувши існуючі мови програмування та після проведення порівняння їх особливостей для розробки програмного забезпечення була обрана скриптова мова програмування Lua за такі її переваги як розширюваність, простота та відносна ефективність. Інструментом для розробки програмного забезпечення був обраний ігровий двигун Solar2D, що як раз підтримує розробку на мові програмування Lua. До інших переваг використання даного інструменту також відносяться його стабільність, ціна та підтримка великого спектру операційних систем як мобільних так і настільних. Після обрання мови програмування та інструментарію стає можливе формування технічного завдання, що повинно скоротити час розробки та сформувати чітке бачення проекту для попередження витрачання зайвого часу на реалізацію непотрібного функціоналу.

РОЗДІЛ 2
АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ «STUDUCK» НА СТАДІЇ ПРОЕКТУ

2.1 Аналіз існуючих архітектурних платформ для розробки програмного забезпечення

Хоча такий термін як «архітектура програмного забезпечення» і є відносно новим для індустрії розробки програмного забезпечення, проте фундаментальні принципи цієї області неумисно використовувались піонерами розробки програмного забезпечення ще починаючи з середини вісімдесятих років двадцятого століття. Проте початком архітектури програмного забезпечення як концепції було покладено ще у 1968 році в науково-дослідницькій роботі Едсгер Дейкстрі та Девідом Парнасу на початку 1970-х. Ці вчені наголошували, що структура системи програмного забезпечення має важливе значення, і що побудова правильної структури – критично важливо. Перші спроби осмислити та пояснити програмну архітектуру системи були повні приблизними формулюваннями та страждали від нестачі організованості, що часто приводило до того, що архітектура виглядала як звичайна діаграма з блоків, з'єднаних лініями. Після цього, у 90-ті роки спостерігались спроби визначення та систематизування бодай основних аспектів для приведення дисципліни у більш цілісний вид. Таким чином Мері Шоу та Девід Герлан написали книгу під назвою «Software Architecture: Perspectives on an Emerging Discipline», що можна перекласти як «Архітектура програмного забезпечення: перспективи нової дисципліни» ще у 1996 році. В ній вони висунули концепції архітектури

програмного забезпечення, такі як компоненти, з'єднувачі, стилі і так далі. З усього цього вийшов перший стандарт програмної архітектури IEEE 1471 відомий як «Рекомендована практика для опису архітектури програмно-інтенсивних систем» котрий прийняли у 2007 році як ISO/IEC 42010:2007. Попри усе це, проектування архітектури програмного забезпечення так і не набула єдиних чітких правил щодо правильного шляху створення систем, через що це все ще може вважатися сумішшю науки та мистецтва.

Для опису архітектури програмного забезпечення використовують мови опису архітектури (Architecture Description Language). Різними організаціями було розроблено кілька різних мов опису архітектури:

- AADL;
- Wright;
- Acme;
- xADL;
- Darwin;
- DAOP-ADL;

Спільними для всіх цих мов елементами є поняття компоненту, з'єднувача та конфігурації. Також, крім спеціалізованих мов, для опису архітектури часто використовується уніфікована мова моделювання UML яка позиціонується як стандарт для моделювання програмних систем і не тільки. Це досягається тим, що досі немає цілковитої згоди у потребах від опису архітектури програмного забезпечення.

Також, окрім різних мов опису, існують різні види архітектури програмного забезпечення по аналогії із різними типами креслень при будівництві різноманітних споруд. В онтології, встановленої ANSI/IEEE 14-71-2000, види є екземплярами точки зору, де точка зору існує для

опису архітектури з точки зору заданої множини зацікавлених осіб. Складається архітектурний опис як правило з двох компонентів: елементи та відносин між ними. В свою чергу, самі види можна поділити на три основні типи:

- модульні – що відображають систему як структуру з різних програмних блоків;
- компоненти і з'єднувачі – що відображають систему як структуру із паралельно запущених компонентів і способу їх взаємодії;
- розміщення – відображає розміщення елементів системи в зовнішніх середовищах.

До модульних видів відносяться:

- вид декомпозиції – що складається з модулів у контексті відношення «є підмодулем»;
- вид використання – що складається з модулів у контексті відносин «використовує»;
- вид рівнів – відображає структуру, в якій пов'язані за функціональністю модулі об'єднані в групи;
- вид класів/узагальнень – відображає класи, які пов'язані між собою відносинами типу «успадковується від» та «є екземпляром».

Компоненти і з'єднувачі часто набувають наступних видів:

- вид процесу – що складається із процесів, з'єднаних операціями комунікації, синхронізації і виключення;
- паралельний вид – складається із компонентів і з'єднувачів, де з'єднувачі уособлюють в собі логічні потоки;
- вид обміну даними – складається із компонентів та з'єднувачів, які створюють, зберігають та отримують постійні дані;

- вид клієнт-сервер – складається із взаємодіючих клієнтів та серверів, а також із таких з’єднувачів між ними як протоколи або ж спільні повідомлення;

Останній тип розміщення зазвичай може бути одним із наступних трьох видів:

- вид розгортання – складається із програмних елементів, їх розміщення на фізичних носіях і комунікаційних елементах;
- вид впровадження – складається із програмних елементів та їх відповідності файловим структурам у різних середовищах;
- вид розподілу роботи – складається з модулів та опису відповідального за впровадження кожного з них.

Розробка та реалізація всіх цих технологій та засобів їх використання має лише одну єдину мету: спрощення. Якісна архітектура це, перш за все, вигідна архітектура, яка зробить процес розробки та супроводу програми більш простими та ефективними. Програму з хорошою архітектурою легше розширювати та змінювати, проводити тестування, налагоджувати та банально розуміти. Із подібних переваг при використанні гарної архітектури можна сформулювати цілком універсальні та конкретні критерії:

- ефективність системи – яка повинна забезпечити надійну роботу коду с виконанням поставлених для нього завдань;
- гнучкість системи – що дасть змогу у майбутньому швидше та легше проводити зміни в існуючому функціоналі;
- можливість розширення системи – що дозволить додавати в систему нові сутності та функції, не порушуючи її основну структуру;

- масштабованість процесу розробки – що дає можливість скорочувати термін розробки за рахунок додавання до проекту нових людей;
- можливість повторного використання – забезпечить полегшення розробки інших систем за рахунок можливого використання готових фрагментів із попередніх систем;
- легкий супровід – який повинен забезпечити відносно легко та швидко розібратися в системі новим людям.

Ну і для більш повної картини можна згадати критерії дизайну поганої архітектури: жорсткість, хрупкість та нерухомість.

Не дивлячись на велике різноманіття критеріїв все ж головною при розробці системи вважається задача зниження складності розробки системи. А для зниження складності поки що нічого окрім роздроблення на частини не було вигадано. Іноді цей принцип називають принципом «розділяй та владарюй», проте по суті справа іде про ієрархічну декомпозицію. Декомпозиція це шлях вирішення однієї великої задачі шляхом вирішення декількох менших, таким чином складна система вимальовується із кількох простіших підсистем, які в свою чергу складаються із ще менших частин и так до тих пір поки частини не будуть досить простими для самого розуміння та створення. На щастя це не лише єдине вірне існуюче рішення, а і універсальне рішення, що вирішує великий спектр задач забезпечуючи пониження складності, воно також забезпечує гнучкість системи, дає зручні можливості для масштабування та покращує стійкість системи. Проте декомпозиція також буває різною. Декомпозицію можна розділити на три типи: ієрархічна, функціональна та High cohesion and Low coupling. Ієрархічну декомпозицію проводять так: спочатку систему розбивають на крупні функціональні модулі або ж

підсистеми, які описують її роботу в цілому. Потім, отримані модулі, аналізуються більш детально і, в свою чергу, діляться на підмодулі або об’єкти. Типовими модулями першого рівня отриманими в результаті першого ділення системи на найбільш крупні складові є «бізнес-логіка», «користувацький інтерфейс», «доступ до БД», «зв'язок з конкретним обладнанням або операційною системою». Функціональна декомпозиція, де система розбивається на підзадачі, які можуть виконуватись незалежно одне від одного. Кожен модуль повинен відповідати за вирішення якоїсь підзадачі та виконувати відповідну їй функцію. Крім функціонального призначення модуль також характеризується набором, необхідних для виконання його функції, даних. Останній тип декомпозиції це High Cohesion and Low Coupling, який коротко можна характеризувати так: модулі отримані в результаті декомпозиції, повинні бути максимально згуртовані всередині (High internal cohesion) та мінімально пов’язані одне з одним (Low external coupling). Згуртованість всередині модуля, говорить про те, що модуль сфокусований на вирішенні однієї вузької проблеми, не займаючись виконанням різнорідних функцій або незв’язаних між собою обов’язків. В свою чергу слабка пов’язаність, означає що модулі, на які розбивається система, повинні бути, за можливості, незалежними або дуже слабко пов’язаними одне з одним. Вони повинні мати можливість взаємодіяти, але при цьому як можна менше знати одне про одного за принципом мінімального знання.

Зрозумівши важливість правильної архітектури та ціну помилки можна почати розробку архітектури освітньої системи «STUDUCK». Після визначення програмних сутностей їх можна розділити на умовні одиниці: сцени. Кожна сцена буде відповідати за певний аспект системи

та зв’язуватись з іншими сценами. Після проведення аналізу наявних сцен їх архітектура має наступний вигляд (рис. 2.1.).

Рис. 2.1. Архітектура сцен додатку

Після подібного розділення можна визначити функціонал кожного окремого блоку програми та виділити увесь функціонал для подальшого ділення його на функціональні частини із яких можна буде вивести системну архітектуру додатку (рис. 2.2.).

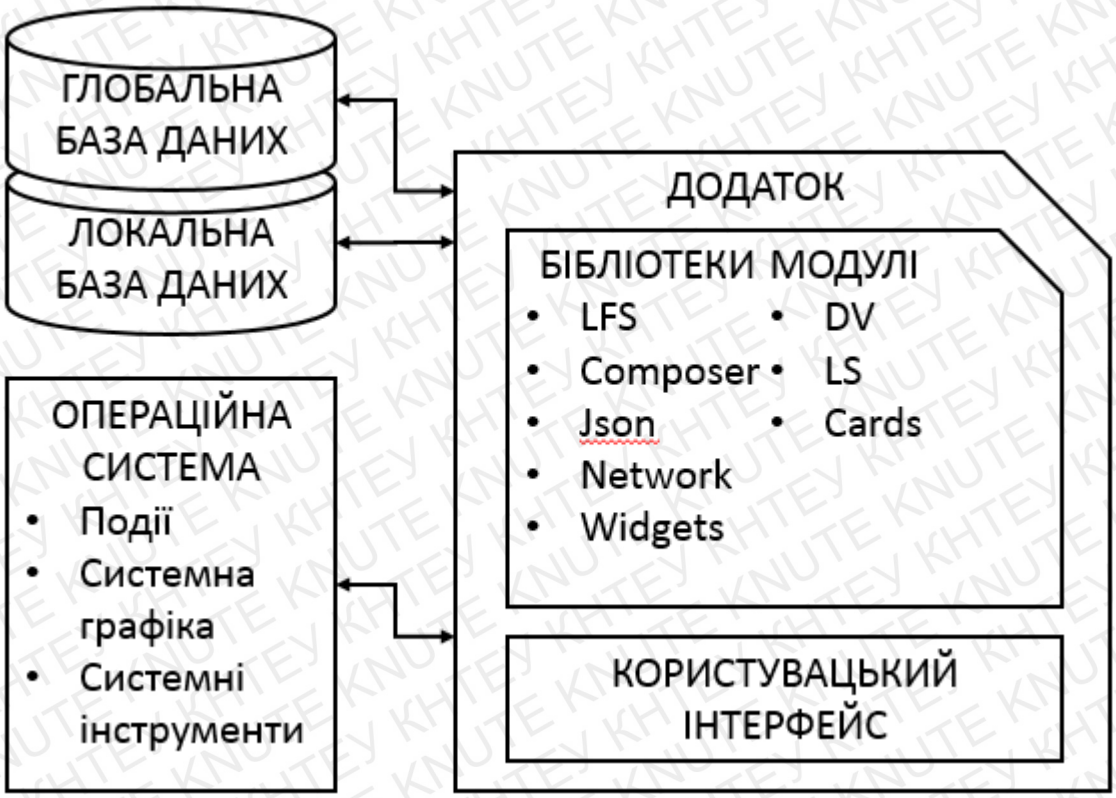


Рис. 2.2. Системна архітектура додатку

2.2 Використання плагінів та модулів під час розробки освітньої системи «STUDUCK»

При розробці програмного забезпечення, будь то мала програма чи велика система, неминуче настає час коли виникає потреба у використанні модулів чи плагінів. Плагін – це незалежно компільований програмний модуль, що динамічно підключається до основної програми та призначений для її розширення або ж використання її можливостей.

Плагіни є досить популярним рішенням як для розробників, так і для користувачів. При розробці не завжди виникає потреба у використанні того чи іншого функціоналу і для того, щоб він не займав місце та не навантажував зайвий раз систему його виносять у плагін, тип самим даючи змогу розробнику використовувати лише необхідний

функціонал та зменшувати підсумковий розмір програми, що є особливо важливим аспектом при використанні програми на мобільних пристроях. Модуль в свою чергу це перш за все функціонально закінчений фрагмент програми. На відміну від плагіну, він використовується переважно лише під час програмування, хоча по своїй суті він дуже схожий на плагін. Саме завдяки використанню модулів можливе спрощення проектування програмного забезпечення та розподілення процесу розробки між групами розробників. При розбивці програмної системи на модулі для кожного модуля вказується функціональність яка ним реалізується а також зв'язок з іншими модулями. Зручність використання модульної архітектури полягає у можливості оновлення модулю без необхідності заміни або налаштування іншої частини системи. Зазвичай модулі оформлюються у вигляді окремого файлу з виконуваним кодом.

Завдяки використанню мови програмування Lua, Solar2Dактивно підтримує як плагіни, так і модульну архітектуру розробки. Завдяки використанню плагінів можливості Solar2Dзначно зростають, так як часто у них використовується нативна мова програмування для окремих операційних систем: java для android, swift або ж Objective-СдляIOSі так далі. Проте зазвичай для плагінів як і для основного коду використовується Lua тобто, плагіни можуть бути трьох видів:

- ті, що складаються лише з Lua коду;
- ті, що складаються із нативної мови C або Java;
- ті, що складаються із Lua та нативного коду на C або Java;

Таким чином при наявності на спеціальному маркетплейсі відповідного плагіну можна отримати майже повний контроль над системою або ж при сильному бажанні навіть написати його самому. Плагіни для Solar2Dрозміщенні в декількох місцях, більшість із них

доступні на окремих площадках: Free Plugins Directory, Solar2D Marketplaceта Solar2D Plugins Marketplace. Для додавання плагіну до проекту достатньо лише додати запис в таблицю plugins у файлі build.settings (рис. 2.3.).

```
-- Plugins section
plugins =
{
    ['plugin.toast'] = {publisherId = 'com.spiralcodestudio'},
    ["plugin.zip"] = {publisherId = "com.coronalabs"},
},
```

Рис. 2.3. Приклад додавання плагіну до проекту

Подібний спосіб підключення забезпечує цілісність проекту так як Solar2Dінтегрує плагін ще на етапі зборки проекту. Використовувати плагін можна в будь якому місці проекту після його завантаження за допомоги стандартної функції Lua require(), наприклад так local toast = require (“plugin.toast”).

Написання свого плагіну також не є важкою задачею якщо використовувати Solar2D і це завдяки дуже добре написаній документації та активній підтримки спільноти. Для створення нового плагіну Lua, Solar2Dлюб’язно представляє пустий шаблон проекту та допоміжні скрипти, які виконають усі необхідні зміни назв файлів та заміну строк як для macOS, так і для Windows. Для цього варто у командному рядку запустити create-project.bat сценарій, замінивши каталог на папку шаблону проекту та вказавши шлях до нової папки проекту з ім’ям плагіну. Після цього варто відредагувати файл metadata.json, що містить інформацію щодо плагіну. Відредагувавши усі необхідні пункти можна переходити до написання власного плагіну (рис. 2.4.).

```lua
-- Створити бібліотеку
local Library = require( "CoronaLibrary" )
local lib = Library:new{ name = "plugin.DemoleculyarizatorEnot", publisherId = "com.PomoiniyYenot" }

-- Створити функцію бібліотеки
lib.destroyAll = function()
    print( "Все знищенно, упс =(" )
end

-- Повернути "lib"
return lib
```

Рис. 2.4. Приклад реалізації плагіну в Solar2D

Через свою локальну схему застосування модулі реалізуються дещо простіше. В простішій формі зовнішній модуль – це просто інший файл типу Lua, який що-небудь повертає, частіше за все, таблицю. Наприклад будова найпростішого модулю (рис. 2.5.) який просто буде виводити інструкцію у консоль та повертати порожню таблицю.

```lua
local M = {}

print( "Модуль завантажений, насолоджуйся" )

return M
```

Рис. 2.5. Будова найпростішого модулю

Подібний модуль мало що може запропонувати у плані функціоналу, тож для розширення його функціоналу варто додати функції, які згодом буде можливо використовувати після підключення модулю до проекту. Для реалізації функції варто лише її прописати як елемент таблиці (рис. 2.6.).

```lua
M.doSomething = function()
    print( "Я НАМАГАЮСЬ!" )
end
```

Рис. 2.6. Реалізація власної функції у модулі

Після завершення реалізації модулю його, як і плагін, варто підключити за допомоги функції require після чого можна використовувати увесь функціонал модулю.

2.3 Висновок до розділу 2

Розробка програмного забезпечення вимагає відповідального ставлення до себе ще на початку проектування. Існує багато різноманітних технологій та методик, що мають спростити як поточний, так і майбутній процес розробки, до них можна віднести проектування архітектури, використання зручних практик як програмування (модульне програмування) так і оформлення робочого процесу в цілому. Активне використання таких простих інструментів як плагіни та модулі повинно значно підвищити гнучкість всієї системі в цілому та спростити тестування програмного забезпечення у майбутньому, а завдяки використанню Solar2Dy купі із мовою програмування Lua ніколи ще не було так просто і зручно їх використовувати та розроблювати як сьогодні.

.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Проектування інтерфейсу додатку

Дизайн мобільних додатків не менш важливий ніж їх функціонал, до того ж саме за ним залишається право першого враження. Саме тому при розробці додатку сааме дизайну варто приділити особливу увагу. Розробка дизайну мобільних додатків завжди бере до уваги не лише естетичну складову, а і те, як зручно буде працювати користувачам з таким додатком на екрані мобільного пристрою. Саме тому, при розробці дизайну, варто звертати увагу та притримуватись таких принципів:

- Дотримуватись достатнього рівня контрасту – що повинно забезпечити гарний рівень візуального сприйняття користувачем тексту та текстових компонентів.
- Відображати основну інформацію на головному екрані – так як основне, що хоче донести до користувача додаток повинно бути видно одразу без переходів та скролів.
- Усі елементи інтерфейсу повинні бути вирівняними – для забезпечення цілісності інтерфейсу та відпрацювання у користувача розуміння інтерфейсу.
- Не варто використовувати чисто чорний колір – для кращого ефекту більш оптимальним буде використання майже чорного – він не буде викликали відчуття мерехтіння через що користувач буде менше втомлюватись та краще сприймати інформацію
- Варто з розумом підбирати розмір шрифту та інтервал поміж

строк – так як маленький шрифт може викликати незручності при читанні а із-за маленького інтервалу все буде зливатись

- Використовувати поєднувані між собою кольори – щоб не відволікати увагу користувача від головного

Дотримання подібних правил повинно максимально ефективно впливати на зручність користування додатком, такі правила допустимі до використання на кожній операційній системі. Проте кожна окрема операційна система має свої шаблони інтерфейсу та правила взаємодії із користувачем які вже добре їм знайомі.

Сьогодні усі ми звикли з тим, що інтерфейси Google усюди виглядають та працюють приблизно однаково, проте так було не завжди. Ще десять років назад, мобільні додатки значно відрізнялись від своїх настільних рішень (рис. 3.1.).

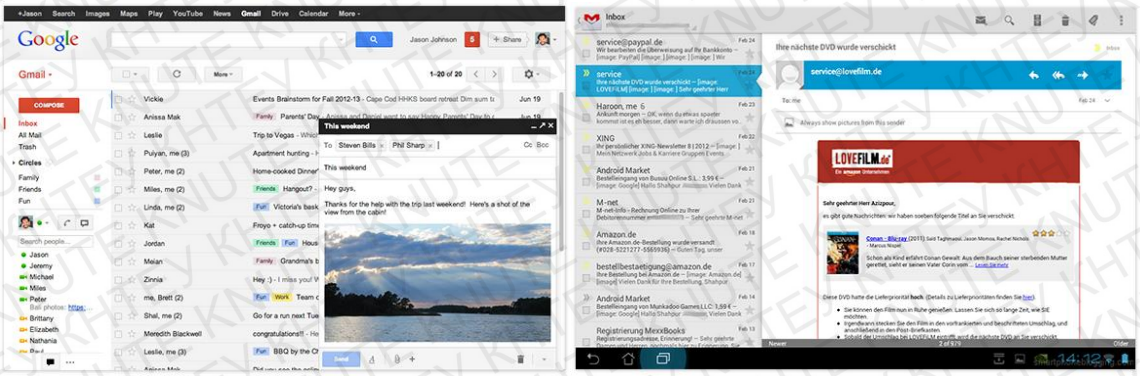


Рис. 3.1. Порівняння інтерфейсу Gmail на комп’ютері та планшеті у 2011 році

Подібна ситуація не могла тривати вічно, саме тому, у 2014 році, в Google вирішили це виправити. Рішенням була розробка єдиного уніфікованого дизайну для усіх програмних рішень яку назвали Material Design. Щодо візуального стилю, Google примирив два таких стилі як скевоморфізм та плаский дизайн додаючи у останній досвід взаємодії з

реальним світом за рахунок знайомих тактильних характеристик та глибини. В основі Material Design лежать чотири принципи:

- Тактильні поверхні – що означає, що всі елементи інтерфейсу – це шари цифрового паперу. Вони розміщуються на різній висоті та генерують тіні. Це допомагає користувачам відрізнити головні елементи від вторинних та робить інтерфейс інтуїтивно зрозумілим;
- Поліграфічний дизайн – зобов'язує підпорядковуватись законам печатного дизайну. Таким чином акцентуючи увагу користувача на потрібному елементі та окреслити ієрархію інтерфейсу;
- Усвідомлена анімація – що підпорядковується одному із найголовніших правил: ніщо ні звідки не береться та нікуди не зникає. Об'єкти дотримуючись цього правила повинні плавно переходити з одного в інший при цьому підказуючи користувачу, як працює інтерфейс;
- Адаптивний дизайн – який повинен дотримуватись усіх вищезазначених правил на пристроях із будь яким розширенням дисплею;

За роки свого існування в Material Design були чітко сформовані компоненти інтерфейсу, до яких відносяться:

- Панелі додатку – бувають двох видів: нижні та верхні. Нижня панель відображає навігацію та ключові дії у нижній частині дисплею. Верхня відображає інформацію та дію, що відноситься до поточного дисплею.
- Фон – який існує позаду усіх інших елементів з контекстом та корисною інформацією.
- Банери – відображають помітні повідомлення та зв'язані з ними додаткові дії.

- Нижня навігація – нижня панель навігації дозволяє переміщатись між основними пунктами призначення в додатку.
- Кнопки – які дозволяють користувачеві виконувати дії та робити вибір одним дотиком;
- Плаваюча кнопка – кнопка зі змінною дією, яка виконує основну дію на поточному екрані.
- Карточки – містять контент та дію по одній темі.
- Таблиці даних – які відображають набір даних.
- Діалоги – інформують користувача о задачі та можуть містити важливу інформацію або ж потребувати прийняття рішення.
- Списки – які можуть бути двох видів. Списки зображень відображають колекцію зображень у організованій сітці, коли класичний список має вид неперервних вертикальних вказівників тексту або зображень.
- Меню – відображає список можливих варіантів на тимчасових поверхнях.
- Панель навігації – забезпечує доступ до пунктів призначення у додатку.
- Вкладки – що впорядковують контент по різним екранам, наборами даних та іншими взаємодіями.
- Та інші.

З подібних елементів можна спроектувати майже будь який інтерфейс додатку. Так як більшість дисплеїв сучасних смартфонів має відношення сторін 2:1 тому, і проектування інтерфейсу буде відбуватись з оглядом на таке відношення сторін. Головний екран додатку (рис. 3.2.) який буде відображати головні події складається із верхньої панелі, на якій знаходиться назва додатку та кнопка для виклику панелі навігації. Під ними знаходиться головна область, що буде складатися із карточок

наповнених інформацію щодо подій. Знизу справа на дисплеї повинна знаходитись плаваюча кнопка, головній функціонал якої це виклик пошуку інформації по додатку.

Рис. 3.2. Прототип дизайну головного екрану мобільного додатку освітньої системи «STUDUCK»

Після виклику панелі навігації головне вікно зсуватиметься вліво, відкриваючи користувачеві доступ до навігації по додатку. Сама панель навігації окрім кнопок для переходу також буде містити елемент з інформацією щодо користувача (рис. 3.3.).

Усі розділи задля збереження однієї стилістики будуть мати майже однаковий вигляд: спільна панель навігації, плаваюча кнопка, верхня панель та карточки з інформацією. Єдиними винятками у цій ідилії є розділи із розкладом та навчальними матеріалами: власне самими матеріалами та тестами для проходження оцінки знань. У будову розділу з розкладом, до звичних елементів додається елемент з вкладками у якому розклад поділяється на тижні: перший та другий (рис. 3.4.). Подібне використання вкладок обумовлено полегшенням основного вікна інтерфейсу.

Рис. 3.3. Прототип дизайну навігаційної панелі додатку

Рис. 3.4. Прототип дизайну інтерфейсу розділу з розкладом

Розділ із навчальними матеріалами від усього додатку буде дещо відрізнятись. Усі вікна у ньому будуть з'єднані послідовно у фіксованому порядку: рубрики, теми, навчальний матеріал, тести (рис. 3.5.).

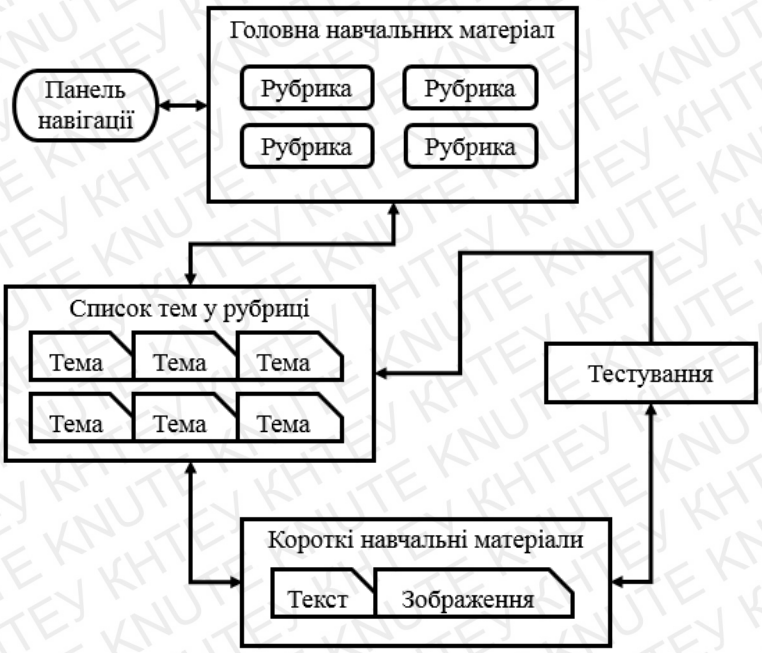


Рис. 3.5. Схема взаємодії вікон у розділі з навчальними матеріалами

Головне вікно із рубриками буде виконане у стилі всього додатку із карточками та панеллю навігації. Проте після вибору рубрики, теми будуть представлятись у вигляді списку тем розміщених у порядку з рекомендацією до їх вивчення (рис. 3.6.).

Після вибору теми, користувач потрапляє до навчального матеріалу, що складається із тексту та зображень. Наприкінці навчальних матеріалів по обраній темі буде розміщена кнопка для переходу у відповідний розділ із тестуванням на обрану тематику. Після проходження тестування користувач отримує оцінку та може спробувати пройти обраний тест ще раз, або ж перейти до наступної теми із обраної рубрики.

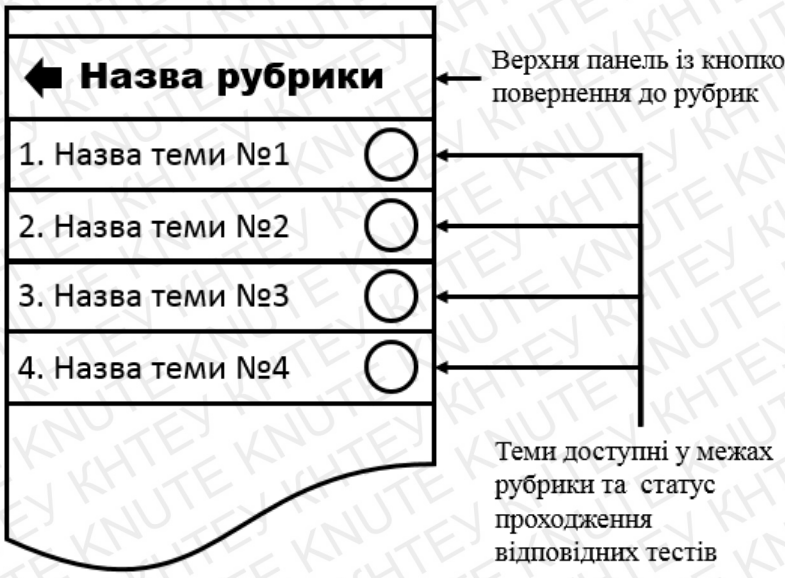


Рис. 3.6. Прототип вікна з темами

3.2 Розробка інтерфейсу мобільного додатку освітньої системи «STUDUCK»

Із сучасним розвитком інструментів для розробки програмного забезпечення реалізовувати інтерфейси стало дуже просто. Так при розробці програмного забезпечення на операційні системи Windows використовуючи інтегроване середовище для розробки Visual Studio достатньо лише переміщати елементи інтерфейсу із списку прямісінько на форму. Аналогічно відбувається розробка інтерфейсу і на операційну систему Android використовуючи Android Studio (рис. 3.7.).

Подібний метод може як подобатись так і дратувати, проте, використовуючи крос-платформні рішення, такі як Solar2D, розробка інтерфейсу кардинально змінюється. В основі такого інтерфейсу, при умові розробки додатку а не гри, стають графічні примітиви: лінії, прямокутники, круги, трикутники та, значно ріже, фігури довільної форми.

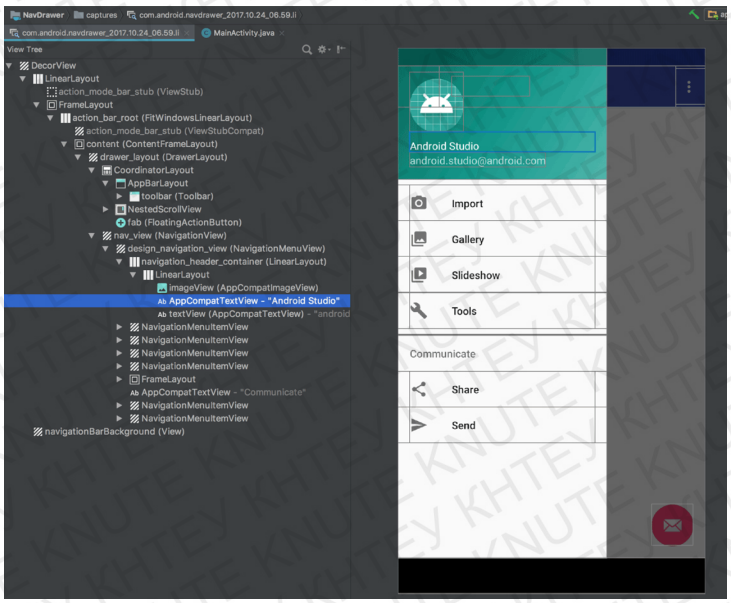


Рис. 3.7. Розробка інтерфейсу у інтегрованому середовищі Android Studio

Solar2D як інструмент створений для розробки програмного забезпечення у форматі 2D звісно ж підтримує більшу частину фігур. Всі вони реалізуються за допомоги вбудованої бібліотеки display. Таким чином, для реалізації верхньої панелі варто застосовувати подібний код (рис. 3.8.)

```
topRect = display.newRect( buttonGroup, C_X, 30*px, _X, 60*px )
topRect:setFillColor( 0, 0.8, 0 )
topRect.name = "topRect"
topRectShadow = display.newRect( buttonGroup, C_X, topRect.y +  topRect.height/2 , _X, 5*px )
topRectShadow.anchorY = 0
topRectShadow.name = "topRectShadow"
local shadowGrad = {
    type = "gradient",
    color1 = {0, 0, 0, 0.5},
    color2 = {0, 0, 0, 0},
    direction = "down"
}
topRectShadow.fill = shadowGrad
sceneName = display.newText( buttonGroup, sceneName, 10*px,
    topRect.y+ topRect.height/4 - 7.5*px, nativeFont, 20*px )
sceneName.anchorX = 0
sceneName.name = "sceneName"
editButton = display.newRect( buttonGroup, _X-20*px, sceneName.y, 40*px, 40*px  )
editButton:setFillColor( 0, 0.8, 0 )
editButtonImg = display.newImageRect( buttonGroup, "res/img/burger_button.png", 20*px, 20*px )
editButtonImg.x, editButtonImg.y = _X-20*px, sceneName.y
editButtonImg.name = "editButton"
editButton:addEventListener( "touch", buttonListener )
```

Рис. 3.8. Основна частина коду для реалізації верхньої панелі

Спершу створюється локальна змінна для кожного елементу. Важливо щоб змінні були локальні, обмежуючи до них доступ із інших фрагментів коду. Після їх створення їм задаються значення. Майже усі об’єкти у Solar2D завдяки мові Lua мають вигляд таблиці із параметрами та їх значеннями. Прикладом такої таблиці може виступати shadowGrad, який містить параметри заливки для тіні верхньої панелі. Таким чином, у змінну topRect записується таблиця із параметрами: батьківська група, положення по X та Y і ширина із висотою. Після цього викликаємо функцію заміни кольору setFillColor передаючи йому об’єкт та вказуючи йому новий колір у форматі RGB та альфа яка не вказується так як немає необхідності міняти прозорість. Далі створюється об’єкт який буде виконувати роль тіні для забезпечення ефекту Material Design у якому виконується додаток, для цього потрібен прямокутник з тією ж шириною, проте куди меншою висотою.

Після створення завдяки параметру fill новому об’єкту передається таблиця із градієнтом shadowGrad в який перефарбовується об’єкт. Для створення підпису поточної сцени використовується команда newText із тієї ж бібліотеки display. Функція приймає у себе наступні параметри: батьківська група, текст який буде відображатись, позиції по X та Y, шрифт та його розмір.

Колір тексту встановлюється аналогічно із прямокутником. І останнім елементом верхньої панелі є кнопка, яка повинна викликати панель навігації. Для її створення використовується зображення burger_button.png яке вказується у виклику функції newImageRect. Для того, щоб кнопка отримала події натискання, до неї застосовується функція addEventListener, де вказується тип взаємодії: tap або touch та функція яка оброблює подію натискання. Результатом виконання подібного коду буде верхня панель (рис. 3.9.) Реалізація карточок

проходить таким же чином, лише замість звичайного прямокутника створюється прямокутник із закругленими кутами завдяки функції newRoundedRect до параметру якого додається радіус закруглення кутів.

Рис. 3.9 Реалізована верхня панель у додатку

Панель навігації є чи не найголовнішим елементом інтерфейсу, бо саме завдяки йому здійснюється переміщення по інтерфейсу додатку і, можливо, саме з ним користувач буде взаємодіяти частіше за все. Ці фактори спонукають відноситись до реалізації панелі навігації більш детально, задля забезпечення максимального комфорту користувача. Панель навігації складається із двох основних елементів: кнопок та бару.

Бар – це верхня частина панелі навігації яка відображає інформацію про користувача. Його використовують також як засіб задля зміщення кнопок донизу, щоб користувачу було простіше користуватись однією рукою і йому не приходилось дотягуватись до верхньої частини дисплею за першою кнопкою. Реалізація бару не сильно важча за реалізацію верхньої панелі. Використовується такий же прямокутник та тінь для нього, єдиною особливістю тут є реалізація аватару користувача та тексту із ім’ям та прізвищем (рис. 3.10.).

Для відображення аватару використовується звичайний круг для створення якого використовується функція newCircle. Так як користувач матиме змогу завантажувати власне фото, яке напевно буде виконане у форматі прямокутника, варто реалізувати обрізку фото, про це подбає параметр об’єкту fill який автоматично генерує маску у об’єкта яка в свою чергу обрізає залишки зображення.

```
avatar = display.newCircle( sceneGroup, _X - 45*px, 60*px , 30*px )

avatar.fill = {type = "image", filename = "avatar.png"}

name = display.newText({
    parent = sceneGroup,
    text = "Sad CAT",
    x = C_X+_X/5/2 - (_X-_X/5)/2.75 ,
    y = avatar.y - avatar.height/5,
    width = (_X-_X/5)-(_X-_X/5)/3 ,
        --height = C_Y,
    font = nativeFont,
    fontSize = 20*px,
    align = "left" } )
name.anchorX=0
```

Рис. 3.10. Реалізація аватару та тексту в барі панелі навігації

Наступною особливістю у барі буде реалізація тексту. У Solar2D існує два види реалізації тексту: прямий та непрямий. Використовуючи прямий метод, існують обмеження у налаштуванні тексту, так, лише використовуючи непрямий метод, доступний такий параметр як align що відповідає за вирівнювання тексту. Завершивши реалізовувати бар, можна переходити до реалізації кнопок (рис. 3.11.) у панелі навігації.

Кожна кнопка у панелі навігації складається із трьох елементів: фон, текст, зображення. Фон найголовніша частина кнопка, так як саме він буде оброблювати події взаємодії. Для створення кнопки використовуються вже знайомі функції newRect, setFillColor, newImageRect, newText. Єдиною особливістю цих кнопок буде наявність у них параметру name.

```
settingsButton = display.newRect( sceneGroup, C_X+_X/5/2, peopleButton.y + peopleButton.height ,
    _X-_X/5, 33*px )
settingsButton:setFillColor( 1, 1, 1 )

    settingsButtonImg = display.newImageRect( sceneGroup, "res/img/settings_button.png",
        25*px, 25*px )
    settingsButtonImg.x, settingsButtonImg.y = settingsButton.x - settingsButton.width/2.75,
        settingsButton.y

    settingsButtonText = display.newText( sceneGroup, lang.settings,
        settingsButtonImg.x + settingsButtonImg.width, settingsButton.y, nativeFont, 20*px )
    settingsButtonText:setFillColor( 0,0,0)
    settingsButtonText.anchorX = 0

    settingsButton.name = "settings"
    settingsButton.tap = goto
    settingsButton:addEventListener( "tap", settingsButton )

leftPart = display.newRect( sceneGroup, _X/5, homeButton.y, _X/5, 33*px )
        leftPart.anchorX = 1
        leftPart:setFillColor( 0.9, 0.9,0.9 )

    if currSceneName == "home" then

        homeButton:setFillColor(0.9,0.9,0.9)
        leftPart.y = homeButton.y

    elseif currSceneName == "news" then
```

Рис. 3.11. Реалізація кнопок в панелі навігації

Створюється це параметр простим додаванням до об’єкту .name. Цей параметр нам необхідний для того, щоб використовувати лише одну функцію обробки натискання для усіх кнопок. Таким чином, функція слухач буде звіряти назву кнопки, та переходити до відповідної сцени. Попри це, у панелі навігації існує ще один елемент: leftPart. LeftPart це звичайний прямокутник, який буде візуальним продовженням кнопки тієї сцени, на якій зараз знаходиться користувач. Для інформування користувача о поточній сцені, кнопка, що відповідає поточній сцені буде зафарбовуватись у сірий колір а зліва від неї буде розташований елемент leftPart (рис. 3.12.). За виконання цієї частини відповідає звичайна перевірка if, яка порівнює назви кнопок із поточною назвою сцени.

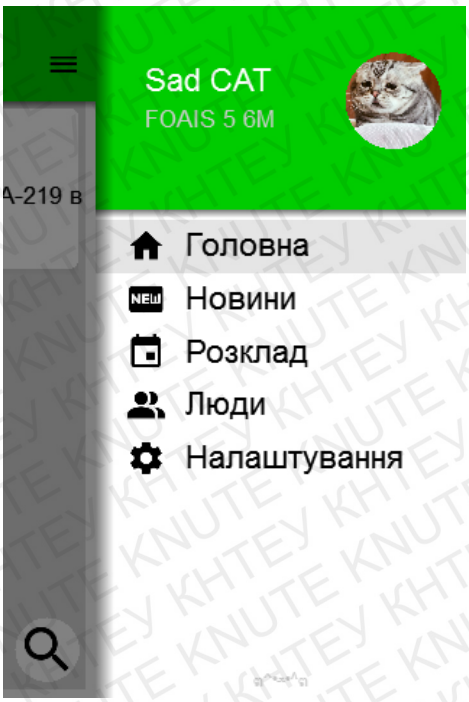


Рис. 3.12. Реалізована панель навігації у додатку

Таким чином використовуючи такі базові геометричні фігури як прямокутники та круги, цілком можливо розробити досить приємний та зручний у використанні інтерфейс при цьому дотримуючись правил дизайну Material Design.

3.3 Розробка функціональної частини мобільного додатку освітньої системи «STUDUCK»

Який би додаток не був гарним, проте без функціоналу користі від нього немає. До основного функціоналу додатку можна винести інформаційну її частину та навчальну. До інформаційної частини відносяться данні о парах, викладачах та новини університету з яких і почнемо.

Так як даних, що будуть використовуватись у інформаційній частині додатку відносно небагато, тож використовувати базу даних буде зайвим, все таки у свій час Lua як раз і розроблювалась як мова для

управління даними. Саме завдяки цьому вона і отримала таку, переважно, табличну форму. Інформація щодо пар зберігається у окремому файлі, який мітить таблицю (рис. 3.13.).

```
local pars =
{
    First =
    {
        Sunday =
        {
            {kabinet = nil, para = nil, prepod = nil},
            {kabinet = nil, para = nil, prepod = nil},
            {kabinet = nil, para = nil, prepod = nil},
            {kabinet = nil, para = nil, prepod = nil},
            {kabinet = nil, para = nil, prepod = nil},
            {kabinet = nil, para = nil, prepod = nil},
            {kabinet = nil, para = nil, prepod = nil}
        },
```

Рис. 3.13. Зразок будови таблиці з розкладом

Таким чином, розклад складається із однієї великої таблиці, що містить на першому рівні дві таблиці призначені для першого та другого тижня відповідно. У кожній із цих таблиць є таблиці днів тижнів, які вже в свою чергу зберігають таблиці із даними щодо пар: кабінет, предмет, викладач. Подібна будова розкладу дозволяє спростити його використання в коді, та прискорити роботу додатку використовуючи рідну для додатку мову (рис. 3.14.).

```
for i = 1, 7 do
    -- print(i)
    firstWeekPars[i] = cardmaker.newCard({Scrollscene, C_X, 55*px + (i-1) *(105*px), _X - 20*px, 100*px})

    firstWeekPars[i].myTexy = display.newText(firstWeekPars[i], days[i],
        firstWeekPars[i][1].x - firstWeekPars[i][1].width/2 + 5*px,
        firstWeekPars[i][1].y - firstWeekPars[i][1].height/2, nativefont, 15*px)
    firstWeekPars[i].myTexy:setFillColor(0,0,0)
    firstWeekPars[i].myTexy.anchorX = 0
    firstWeekPars[i].myTexy.anchorY = 0
    local x = 0
    for w = 1, 7 do

        if pars[currentWeek][days[i]][w].prepod ~= nil then

            para = display.newText( firstWeekPars[i], pars[currentWeek][days[i]][w].para, firstWeekPars[i][1].x,
                firstWeekPars[i].myTexy.y + firstWeekPars[i].myTexy.height*1.5 +  x*(15*px), nativefont, 15*px )
            para:setFillColor( 0,0,0 )

            x = x+1
        end
    end

end
```

Рис. 3.14. Зразок роботи із таблицею розкладу

Інформація про викладачів оброблюється подібним чином, проте цікавим є можливість виклику сторонніх додатків. Таким чином використовуючи мітку «mailto:» у функції system.openURL можна відкрити поштовий клієнт на пристрої передавши йому необхідну нам пошту, або ж телефон якщо використовувати мітку «tel:» таким чином спрощуючи зв'язок між студентом та викладачем.

Новини мають властивість постійно оновлюватись, тож їх реалізувати використовуючи звичайну таблицю не вийде. Для отримання останніх університетських новин було вирішено використовувати інтернет сторінку університету з новинами. Для такого методу необхідно використовувати парсер сторінок. Тож після додавання парсеру функцією require у виконуваний файл, варто отримати код сторінки, що можна зробити скачавши її як файл із розширенням .html.

Після отримання файлу можна проводити парсинг. Для цього варто завчасно визначити які елементи на сторінці нам необхідні. Це можна зробити використовуючи консоль у браузері та виділивши необхідний елемент.

Визначившись із елементами, ми додаємо їх у пошук і виконуємо його. Після закінчення пошуку, парсер повертає нам таблицю із результатами які варто записати у спільну таблицю із відповідними назвами елементів для спрощеного використання у майбутньому (рис. 3.15.).

Після чого заповнюємо карточки із отриманим контентом завантажуючи зображення за посиланнями отриманими в результаті парсингу (рис. 3.16.).

```
local root = htmlparser.parse(contents, 3000)

local sel = root("div.nnews_item span[class]")
local imglink = root("div.nnews_item img")
local date = root("div.nnews_item small[style]")

for e in pairs(sel) do

    --news table all
    news[e] = {}
    --text
    news[e]["txt"] = sel[e]:getcontent()
    --imagine link
    news[e]["imglink"] = imglink[e].attributes["src"]
    --date of publication
    news[e]["date"] = date[e]:getcontent()
    -- link on news (add knute.edu. before go)
    news[e]["link"] = sel[e].parent.attributes["href"]

end
```

Рис. 3.15. Робота із парсером

Як згадувалось вище, інформаційна частина додатку не єдина в додатку, також є навчальна частина. Навчальна частина будується навчальних матеріалах, які реалізовані у вигляді вже знайомих Lua таблиць що містять елементи із параметрами type та content (рис. 3.17.).

Рис. 3.16. Результат використання парсеру у вікні новин

```
local DBL1 =
{
    {type = txt, content = "База даних (англ. database) -
    {type = limg, content = "learn/db/l1/1/png"},
    {type = gimg, content = "https://upload.wikimedia.org/
    {type = txt, content = "В загальному випадку базою да
```

Рис. 3.17. Приклад будови навчального матеріалу

Параметр type відповідає за тип контенту, при відображенні навчального матеріалу, програма звіряється із типом даних та виконує відповідні інструкції:

- txt – тип тексту, при якому програма створює звичайний новий об’єкт типу newText та заповнює його параметром content;
- limg – тип локального зображення, при якому програма створює посередині об’єкт типу newImageRect та заповнює його зображенням, що знаходиться за вказаним у параметрі content шляхом. При відсутності зображення за вказаним шляхом завантажується стандартне зображення із помилкою»
- gimg – тип глобального зображення, що знаходиться в інтернеті. При визначенні такого типу даних програма створює об’єкт типу newImageRect та намагається завантажити зображення за посиланням що знаходиться у параметрі content, після чого заповнює отриманим результатом об’єкт.

Така проста будова навчальних матеріалів повинна максимально зменшити складність їх реалізації, а відсутність перенавантаженого інтерфейсу дозволить користувачеві не відволікатись що спростить та покращить процес навчання.

В свою чергу, тести реалізуються також доволі просто. Створюється таблиця формату Lua яка містить у собі таблицю із двома елементами: запитання та таблиця із чотирьох елементів де зберігаються правильні відповіді. Головною умовою є знаходження правильної

відповіді на першій позиції. В свою чергу, додаток розташує їх у випадковому порядку. Усі тести мають лише чотири варіанти відповіді, серед яких лише один варіант є правильним.

3.4 Висновки до розділу 3

У даному розділі був розроблений та частково розглянуті засоби реалізації розробленого інтерфейсу. Після чого була розроблена функціональна частина додатку. Розроблений базовий комплекс мобільного додатку освітньої системи з можливістю подальшої модифікації та розширення у області як навчання так і інформації. Обраний інструмент в процесі використання під час розробки повністю доказав свою функціональність та великий можливий спектр застосування, що по праву робить його одним із найкращих рішень на ринку.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В ході виконання випускного кваліфікаційного проекту було розглянуто питання поліпшення освітнього процесу завдяки використанню сучасних технологій та проаналізовані багато можливих засобів та можливих інструментів для створення програмних додатків для різноманітних цільових платформ, що дозволило сформувати технічне завдання на розробку програмного забезпечення. Завдяки використанню крос-платформного інструменту для розробки Solar2D, що використовує під час розробки мову програмування Lua, була розроблена мобільна версія додатку для освітньої системи «STUDUCK».

Було розроблено структуру та архітектуру мобільного додатку із активним застосуванням принципу декомпозиції. Додаток виконаний у популярному на момент розробки дизайні мобільних додатків Material Design, який завдяки своїй простоті розуміння та розробки з кожним роком поступово охоплює і такі стаціонарні платформи як Windows та Linux.

У подальшому розвитку додатку крайнє необхідне більше використання хмарних технологій для зберігання та обробки більшості даних, тісна інтеграція із даними університету, додавання інших мов інтерфейсу та побудова і розробка додаткових програмних рішень для створення та обробки внутрішніх даних формату Lua із подальшим розширенням навчальних матеріалів та підняттям їх якості використовуючи більш сучасні методи представлення інформації такі як відео, інтерактивні об’єкти тощо.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ian Dees. Lua // Seven More Languages in Seven Weeks. Languages That Are Shaping the Future / Bruce Tate, Fred Daoud, Jack Moffitt, Ian Dees. – The Pragmatic Bookshelf, 2015. – С. 1–48. – 320 с. – ISBN 978-1941222157.
2. Roberto Ierusalimschy. Programming in Lua. – 3-nd ed.. – 2012. – ISBN 9788590379850.
3. Kurt Jung, Aaron Brown. Beginning Lua Programming. — John Wiley & Sons, 2011. — 675 с. — (Programmer to Programmer). — ISBN 9781118079119.
4. Mário Kašuba. Lua Game Development Cookbook. — Packt Publishing, 2015. — 402 с. — ISBN 978-1849515504.
5. David Young. Learning Game AI Programming with Lua. — Packt Publishing, 2014. — 352 с. — ISBN 978-1783281336.
6. Jayant Varma. Learn Lua for iOS Game Development. — Apress, 2012. — 410 с. — ISBN 9781430246626.
7. Федерико Бьянкуцци, Шейн Уорден. Глава 7. Lua // Пионеры программирования. Диалоги с создателями наиболее популярных языков программирования = Masterminds of Programming: Conversations with the Creators of Major Programming Languages. – Символ, 2011. – С. 211–230. – 608 с. – 1500 экз. – ISBN 9785932861707.
8. Paul Clements; Felix Bachmann; Len Bass; David Garlan; James Ivers; Reed Little; Paulo Merson; Robert Nord; Judith Stafford. Documenting

9. Software Architectures: Views and Beyond. — Second Edition. — Addison-Wesley Professional, 2010. — ISBN 978-0-13-248861
10. Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice, Third Edition. Addison Wesley, 2012, ISBN 978-0321815736

Інтернет ресурси:

1. Офіційна документація Solar2D. [Електронний ресурс] – Режим доступу: https://docs.coronalabs.com
2. Офіційна документація Material Design [Електронний ресурс] – Режим доступу: https://material.io/design

ТЕХНІЧНЕ ЗАВДАННЯ

Розроблювальний проект у кінці має відповідати заданим вимогам серед яких основними є надання користувачу важливої інформації щодо процесу навчання: розклад занять, розклад дзвінків, інформацію щодо викладачів, важливі новини та навчальні матеріали з можливістю перевірки засвоєного матеріалу. Освітня система має бути реалізована на операційну систему Android з подальшою можливістю швидкої реалізацією на інші мобільні платформи такі як IOS та Windows Phone. Мінімальна підтримувальна версія операційної системи додатком Android 4.4 та вище. Мова користувацького інтерфейсу повинна бути реалізована англійською та українською мовами в залежності від вибору користувача. Повинен бути реалізований простий шлях для додавання у подальшому нових локалізацій. Інтерфейс додатку повинен бути реалізований у стилістиці Material design для вертикального відображення. Складається проект із наступних вікон:

- Головний екран (дім);
- Екран з новинами;
- Екран з розкладом;
- Екран з викладачами;
- Екран з навчальними матеріалами;
- Екран з налаштуваннями додатку.

Мета головного екрану надання короткої необхідної інформації користувачу при запуску додатку таких як:

- Яке наразі іде заняття, де воно проходить, хто його веде та скільки до його кінця або у випадку відсутності заняття вказати інформацію щодо наступного;
- Останні важливі новини;
- Останні розглянуті викладачі з можливістю швидкого зв’язку.

Вікно з новинами повинно відображати усі останні новини навчального закладу в форматі заголовків із можливістю швидкого відкриття вибраної новини у сторонньому браузері.

Вікно розкладу повинно мати компактний вигляд у якому відображається уся необхідна інформація поточного дня, або наступного у разі відсутності занять у поточному дні. Повинен бути реалізований швидкий перехід між парними та непарними тижнями при умові їх наявності.

Вікно з контактами викладачів повинна відображати усіх викладачів, що мають зв'язок із користувачем. Елемент з викладачем повинен надавати швидкий доступ до текстового та мобільного зв’язку з викладачем. При потребі повинна бути відображена уся необхідна додаткова інформація щодо викладача.

Навчальні матеріали повинні бути розподіленні в залежності від предмету. Всередині кожного предмету розподілені по темам. В кінці кожного навчального матеріалу повинен бути реалізований швидкий доступ до тестування, що відповідає тематиці навчального матеріалу. Також доступ до тестів повинен бути відкритий у вікні із самими навчальними матеріалами.

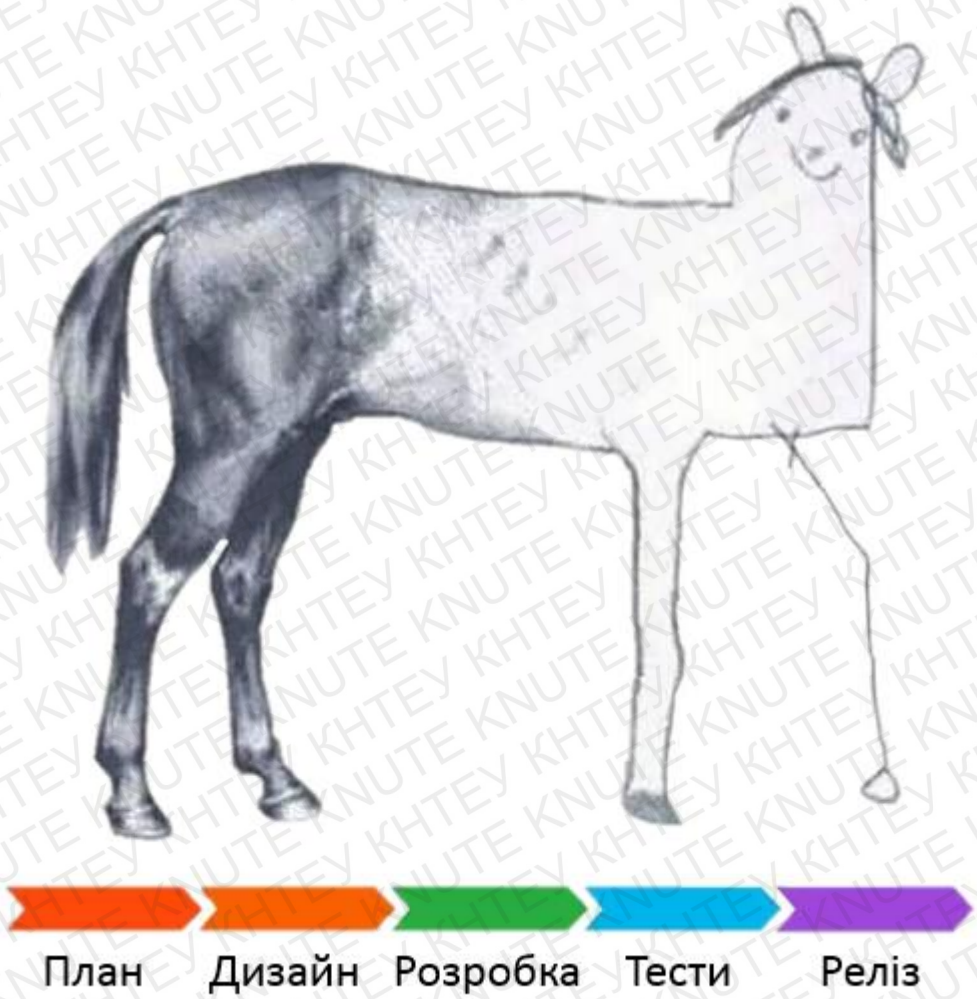
Останнє вікно мітить налаштування додатку. Він має включати в себе налаштування профілю студента та всієї супутньої інформації.

Саме у вікні налаштувань користувач матиме змогу змінити мову додатку на вподобану йому.

ДОДАТКИ

Додаток А

Життєвий цикл проекту та вплив його етапів на стан проекту



Додаток Б

Лістинг коду головного вікна додатку

```
local composer = require( "composer" )

local ls = require("loadsave")
local settings = ls.loadTable("settings.json")
local batons = require("res.scripts.batons.batons")
local win = require("res.scripts.windowsMaker.windowsMaker")
local konstruktor = require("res.scripts.karkas")

local cardmaker = require("res.scripts.cards")

local widget    = require("widget")

local scene = composer.newScene()

local pars = require("res.scripts.pari")

local weeksRect
local topRect
local currWeekLine

local firstWeekScroll
local secondWeekScroll


local function goToFirst()
   transition.to( currWeekLine, {time=200, x = C_X-C_X/2} )

        transition.to( firstWeekScroll, {time=200, x = C_X} )
        transition.to( secondWeekScroll, {time=200, x =  C_X*3} )
end

local function goToSecond()
    transition.to( currWeekLine, {time=200, x = C_X+C_X/2} )

        transition.to( firstWeekScroll, {time=200, x = -C_X} )
        transition.to( secondWeekScroll, {time=200, x = C_X} )
```

```
end


--[[
local previousY
local previousX
local function touchRectListener(event)
    local phase = event.phase

    if ( event.phase == "began" ) then
        -- Code executed when the button is touched
        startx = event.x
        starty = event.y
        previousY = event.y
        previousX = event.x
        --print( "object touched = " .. tostring(event.target) )  -- "event.target" is the
touched object
    elseif ( event.phase == "moved" ) then

        if ((event.x - previousX) < (event.y - previousY)) then
            print("left")
        end

        if event.y < previousY then
            transition.to( weeksRect, {time = 500, y = topRect.y } )
        elseif event.y > previousY then
            transition.to( weeksRect, {time = 500, y = topRect.y +
topRect.height/2+25*px } )
        end
        print("P " ..  previousY .."\nC " .. event.y)
        previousY = event.y
        previousX = event.x

        -- Code executed when the touch is moved over the object
        --print( "touch location in content coordinates = " .. event.x .. "," .. event.y )
    elseif ( event.phase == "ended" ) then
        -- Code executed when the touch lifts off the object
        --print( "touch ended on object " .. tostring(event.target) )
    end
    return true
```

```
end
--]]

local function scrollListener( event )

    local phase = event.phase
    if ( phase == "began" ) then print( "Scroll view was touched" )
    elseif ( phase == "moved" ) then print( "Scroll view was moved" )
        -- проверять в какую сторону мувед и если вверх то вывернуть недели
если внис то свернуть
    elseif ( phase == "ended" ) then print( "Scroll view was released" )
    end

    -- In the event a scroll limit is reached...
    if ( event.limitReached ) then
        if ( event.direction == "up" ) then
            print( "Reached bottom limit" )


        elseif ( event.direction == "down" ) then
            print( "Reached top limit" )
        elseif ( event.direction == "left" ) then print( "Reached right limit" )
            goToSecond()

        elseif ( event.direction == "right" ) then print( "Reached left limit" )
            goToFirst()
        end
    end

    return true
end


-- -----------------------------------------------------------------------------------
-- Scene event functions
-- -----------------------------------------------------------------------------------

-- create()
function scene:create( event )
```

```
    local sceneGroup = self.view
    Scrollscene = konstruktor.buildKarkas({mother = sceneGroup, sceneName =
"schedule", scrollListener = scrollListener, horizontalOff = true})
--[[
    local touchRect = display.newRect( sceneGroup, C_X, C_Y+C_Y/3, _X, _Y-
C_Y/3 )
    touchRect:setFillColor( 0.4, 0,0, 0.3 )
    touchRect:addEventListener( "touch", touchRectListener )
--]]

    local topRectShadow
    local sceneName
    local editButton
    local darkSide
    local bg

    for i = Scrollscene.parent.numChildren, 1, -1 do
        local currObj = Scrollscene.parent[i]
        print(tostring(Scrollscene.parent[i].name))
        if currObj.name == "topRectShadow" then
            topRectShadow = currObj
            --currObj:removeSelf() -- не удалить, а переместить тень ниже
            --currObj=nil
        elseif currObj.name == "topRect" then
            topRect = currObj
        elseif currObj.name == "sceneName" then
            sceneName = currObj
        elseif currObj.name == "editButton" then
            editButton = currObj
        elseif currObj.name == "scrollView" then
            firstWeekScroll = currObj
        elseif currObj.name == "darkSide" then
            darkSide = currObj
        elseif currObj.name == "bg" then
            bg = currObj
        end
    end


secondWeekScroll = widget.newScrollView( --Главный
    {
```

```
        --buttonGroup,
        top = 60*px,
        left = 5*px + _X,
        width = _X-10*px,
        height = _Y - 65*px ,
        --scrollWidth = 0,
        --scrollHeight = 0,
        horizontalScrollDisabled = false,
        listener = scrollListener,
        --bottomPadding = -1025,
       -- rightPadding = -240,
        backgroundColor = { 0.85,  0.85,  0.85 }

    }
    )

    scrollView.name = "scrollView"

Scrollscene.parent:insert(secondWeekScroll )
secondWeekScroll:toBack( )


    weeksRect = display.newRect( topRect.parent, C_X, topRect.y +
topRect.height/2+25*px , _X, 50*px )
    topRectShadow.y = weeksRect.y +  weeksRect.height/2
    weeksRect:setFillColor( 0,0.8,0 )


    local firstWeekFon = display.newRect( topRect.parent, C_X - C_X/2,
weeksRect.y, C_X, 50*px )
    firstWeekFon:setFillColor( 0,0.8,0 )
    firstWeekFon:addEventListener( "tap", goToFirst )
    local firstWeekText = display.newText( topRect.parent, lang.firstWeek, C_X
- C_X/2, weeksRect.y, nativeFont, 18*px)

    local secondWeekFon = display.newRect( topRect.parent, C_X + C_X/2,
weeksRect.y, C_X, 50*px )
    secondWeekFon:setFillColor( 0,0.8,0 )
    secondWeekFon:addEventListener( "tap", goToSecond )
    local secondWeekText = display.newText( topRect.parent, lang.secondWeek,
C_X + C_X/2, weeksRect.y, nativeFont, 18*px)
```

```
    currWeekLine = display.newRect( topRect.parent, C_X-C_X/2, weeksRect.y
+ weeksRect.height/2-2.5*px , C_X, 5*px )


    topRect:toFront( )
    sceneName:toFront( )
    editButton:toFront( )
    darkSide:toFront( )
    bg:toBack( )


---[[
    local days = {
    "Sunday",
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday",
}
--[[
for i = 1, #days do

        if days[i] == os.date( "%A" ) then
            today = checkAll(i)
        end

    end
]]

local firstWeekPars = {}
local secondWeekPars = {}

    local currentWeek = "First"

    for i = 1, 7 do
      -- print(i)
        firstWeekPars[i] = cardmaker.newCard({Scrollscene, C_X, 55*px + (i-1)
*(105*px), _X - 20*px, 100*px})
```

```
      firstWeekPars[i].myTexy = display.newText(firstWeekPars[i], days[i],
         firstWeekPars[i][1].x - firstWeekPars[i][1].width/2 + 5*px,
         firstWeekPars[i][1].y - firstWeekPars[i][1].height/2, nativefont, 15*px)
      firstWeekPars[i].myTexy:setFillColor(0,0,0)
      firstWeekPars[i].myTexy.anchorX = 0
      firstWeekPars[i].myTexy.anchorY = 0
      local x = 0
      for w = 1, 7 do

         if pars[currentWeek][days[i]][w].prepod ~= nil then

            para = display.newText( firstWeekPars[i],
pars[currentWeek][days[i]][w].para, firstWeekPars[i][1].x,
               firstWeekPars[i].myTexy.y + firstWeekPars[i].myTexy.height*1.5
+  x*(15*px), nativefont, 15*px )
            para:setFillColor( 0,0,0 )

            x = x+1
         end
      end

   end

   for i = 1, 7 do
      -- print(i)
      firstWeekPars[i] = cardmaker.newCard({secondWeekScroll, C_X, 55*px +
(i-1) *(105*px), _X - 20*px, 100*px})

      firstWeekPars[i].myTexy = display.newText(firstWeekPars[i], days[i],
firstWeekPars[i][1].x - firstWeekPars[i][1].width/2 + 5*px  ,
firstWeekPars[i][1].y - firstWeekPars[i][1].height/2, nativefont, 15*px)
      firstWeekPars[i].myTexy:setFillColor(0,0,0)
      firstWeekPars[i].myTexy.anchorX = 0
      firstWeekPars[i].myTexy.anchorY = 0
      local x = 0
      for w = 1, 7 do

         if pars["Second"][days[i]][w].prepod ~= nil then
```

```
            para = display.newText( firstWeekPars[i],
pars["Second"][days[i]][w].para, firstWeekPars[i][1].x,
firstWeekPars[i].myTexy.y + firstWeekPars[i].myTexy.height*1.5 +
x*(15*px), nativefont, 15*px )
            para:setFillColor( 0,0,0 )

            x = x+1
        end
    end

end

end


-- show()
function scene:show( event )

    local sceneGroup = self.view
    local phase = event.phase

    if ( phase == "will" ) then
        -- Code here runs when the scene is still off screen (but is about to come on
screen)

    elseif ( phase == "did" ) then
        -- Code here runs when the scene is entirely on screen

    end
end


-- hide()
function scene:hide( event )

    local sceneGroup = self.view
    local phase = event.phase

    if ( phase == "will" ) then
        -- Code here runs when the scene is on screen (but is about to go off screen)
```

```
    elseif ( phase == "did" ) then
        -- Code here runs immediately after the scene goes entirely off screen
        composer.removeScene( "scenes.schedule" )

    end
end


-- destroy()
function scene:destroy( event )

    local sceneGroup = self.view
    -- Code here runs prior to the removal of scene's view

end


-- -----------------------------------------------------------------------------------
-- Scene event function listeners
-- -----------------------------------------------------------------------------------
scene:addEventListener( "create", scene )
scene:addEventListener( "show", scene )
scene:addEventListener( "hide", scene )
scene:addEventListener( "destroy", scene )
-- -----------------------------------------------------------------------------------

return scene
```