Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

Розробка крос-платформного навчального додатку «PuzzLearn»

Студента 4 курсу, 7 групи,
напряму підготовки 6. 050103
«Програмна інженерія» _____ Струка Владислава Сергійовича
підпис студента

Науковий керівник
кандидат технічних наук,
доцент _____ Цензура Микола Олександрович
підпис керівника

Гарант освітньої програми
кандидат технічних наук,
доцент _____ Цензура Микола Олександрович
підпис керівника

КИЇВ – 2019

ЗМІСТ

ВСТУП

Майбутнє за технологіями. Це твердження вже ні для кого не є секретом. С кожним днем технології розвиваються та покращуються, їх сфера застосування розростається з неймовірними темпами, що ще вчора здавалось фантастикою сьогодні вже є реальністю: вантажівки вже їздять без водіїв, каси можуть працювати без касирів, а кількість працівників на заводах зводиться до мінімуму. У мріях минулих поколінь це все значило б що люди могли б безтурботно відпочивати, поки технології роблять їх справу, але жорстока реальність трактує свої правила: безробіття зростає і люди живуть у бідності. Таким чином ручний труд знецінюється на фоні чого все більше зростає цінність інтелектуальна. На щастя люди є розумними створіннями, що дає їй можливість навчатися та розвиватись вивчаючи нові сфери діяльності, як приклад ті самі інформаційні технології.

Одним із найперспективніших розділів інформаційних технологій є програмування. Багато людей вивчають різноманітні мови на кшталт англійської, німецької чи італійської при цьому зовсім нехтуючи можливістю вивчення мов програмування. Людям, що живуть у нашому, часі досить сильно пощастило, вони живуть у час Інтернету, у часі в якому майже вся інформація світу доступна будь де при умові наявності там Інтернету звісно. Також на відміну від часу, коли інформаційні технології тільки зароджувались, зараз існує великий вибір як спеціальностей, так і інструментарії для роботи у цих спеціальностях. Лише мов програмування існує більше ніж дві з половиною тисячі, що вже казати про можливість написання своєї мови. Мови різні, та різні за

складністю освоєння. Для багатьох мов вже створені навчальні матеріали, а для мов лідерів, таких як Java, Python, C# та інших, матеріалів на стільки багато, що жодній людині і життя не вистачить щоб ознайомитись з всім. При цьому такий зручний інструмент як Corona SDK, у якому ведеться розробка на мові програмування Lua, має мінімальну кількість зручних навчальних матеріалів, особливо у вигляді прогрманого додатку. Таким чином мною було вирішено розробити на “Corona SDK” навчальний мобільний додаток, що буде містити навчальний курс с розробки програмного додатку у “Corona SDK” за допомоги мови програмування Lua.

Об’єкт дослідження: процес навчання.

Предмет дослідження: процес розробки мобільного додатку у “Corona SDK” з допомоги мова програмування Lua

Мета виконання дипломної роботи: розробка програмного забезпечення мовою програмування Lua за допомоги Software Development Kit під назвою «Corona SDK».

Завдання: дослідити предметну область тестування у навчальному процесі; проаналізувати наявні інструменти для розробки крос-платформного програмного забезпечення та порівняти мови програмування, що дозволяють вести крос-платформну розробку; сформувати технічне завдання на розробку навчального додатку «PazzLearn»; розробити архітектуру додатку; спроектувати користувацький інтерфейс додатку.

РОЗДІЛ 1. ВІДОМОСТІ ЩОДО ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ

1.1 Тестування у навчальному процесі та типи тестів

При процесі навчання найбільшу роль грає саме перевірка отриманих знань. Бо саме після перевірки можна тверезо оцінити отримані знання, або дізнатися які саме матеріали варто повторити для досягнення кращого результату. Майже завжди для цього використовують тестування тих, чи інших типів. За час свого розвитку з'явилося багато різноманітних способів тестування, до них можна віднести такі як:

а) Бінарний вибір. Уособлює тест, коли у якості відповіді на питання існує два варіанти: «так» або «ні»

б) Вибір «один із багатьох». Вибір одного істинного варіанту відповіді із декількох запропонованих

в) Заповнення бланку. Тип тестування, де користувач повинен ввести у текстове поле відповідь у вільній формі, але з дотриманням певних правил конкретного тесту.

г) Множинний вибір. Тестування у якому потрібно вибрати усі правильні відповіді із багатьох запропонованих.

д) Маніпулювання об'єктами. Тестування, в якому варто переміщувати запропоновані об'єкти у залежності із заданими правилами.

Саме останній варіант з маніпулюванням об'єктами підходить як найкраще, бо це виключає можливість звичайного бездумного

вгадування відповіді, та дає більшу можливість саме навчити користувача, а не сварити його за помилки, при не правильному виборі.

1.2 Мова програмування Lua

На сьогоднішній день існує багато можливих способів написання програмного додатку на мобільні платформи, серед них офіційні середовища розробки, або кросплатформні рішення із використання сторонніх мов програмування. Однією із розповсюджених мов програмування у кросплатформному середовищі є Lua.

Lua – це перш за все процедурний динамічно типізована модульна мова з автоматичним управлінням пам'яттю. Включає в себе базові елементи для підтримки функціонального і об'єктного стилів програмування. Таким чином, Lua можна називати мультипарадигменою мовою. Так як основним призначенням Lua є вбудовування, вона має ефективні засоби міжмовної взаємодії, орієнтовані, головним чином, на виклик бібліотек C і на роботу в C-оточенні.

Мова підтримує невелику кількість вбудованих типів даних: логічні значення, числа, рядки, функції, потоки. Типові комбіновані структури даних, такі як масиви, набори, списки і записи, відсутні, замість всіх них використовується одна базова структура Lua – таблиця. Функції в Lua є об'єктами першого класу, можуть присвоюватися і передаватися в параметрах. Підтримуються замикання, є можливість створення функцій вищих порядків. Об'єктна система прототипна, відсутня явна підтримка спадкування, однак вона легко реалізується за допомогою метатаблиць.

Взагалі, Lua прагне забезпечити гнучкі метафункції, які можуть бути розширені в міру необхідності, а не постачати набір функцій, специфічних для конкретної парадигми програмування. Як результат,

основа мови проста і легко адаптується до більшості додатків. Надаючи мінімальний набір базових засобів, Lua намагається знайти баланс між потужністю і розміром, що їй в певній мірі і вдається.

1.3 Corona SDK

Corona SDK – це набір із засобів розробки, утиліт та документації, що дозволяють програмістам створювати програмне забезпечення. Розроблена Corona SDK була в середині 2009 року командою Corona Labs inc. На сьогоднішній день програми написані завдяки Corona SDK підтримуються на неймовірній кількості платформ таких як Mac OS, Windows, IPhone/IPad, tvOS, Android, Kindle Fire, Android TV. Розробку можна вести на операційних системах Windows та Mac OS, а з недавнього часу до цього списку приєднався Linux.

Corona використовує інтегровану мову програмування Lua. Програмне забезпечення має два режими роботи: Corona Simulator та Corona Native. Corona SDK має великий спектр налаштувань та розширення що досягається завдяки великій кількості наявних плагінів та готових рішень. Не зважаючи на це стандартний пакет API Corona має великий можливий функціонал для роботи з аудіо та графікою, криптографією, мережевою інформацією та інформацією про дані вбудованих у пристрій показників такі як акселерометр, GPS, інформацію про пристрій та власні віджети і навіть ефекти часточок.

Головним досягненням Corona SDK є можливість проводити тестування «на льоту». Будь які зміни коду миттєво відображаються у симуляторі, а при умові використання Corona Live Server, всі зміни можливо перевірити одразу на самих пристроях, різних розмірів та форм факторів. На ринку безкоштовних кросплатформних рішень Corona SDK явно займає лідируючі позиції яких він і заслуговує.

1.4 Формування технічного завдання

Основною метою проекту є розробка мобільного додатку, який міститиме навчальний курс та збірник завдань по збиранню елементів коду із вже наявних строчок. Додаток повинен працювати на мобільних пристроях, що працюють під операційною системою Android. Мова інтерфейсу додатку: англійська. Розроблений додаток повинен бути сумісний з пристроями, що працюють на версії операційної системи Android 4.4.4 та вище. Вся інформація з навчального курсу та завдань повинні зберігатися у базі даних, яка буде розміщуватись у інтернеті. Загальний дизайн додатку повинен бути виконаний у єдиній стилістиці побудованої на основі Material design. Зовнішній вигляд кожного із екранів повинні бути розроблені лише під вертикальну орієнтацію із неможливістю її змінити. Проект має складатися із таких екранів:

а) Головний екран
б) Екран з уроком
в) Екран з завданням
г) Екран із пазлами що створив користувач
д) Екран створення пазлів

Головний екран має надавати користувачу вибір із основних можливостей:

а) Перевірка обновлення бази даних
б) Вибір уроку для його проходження
в) Вибір завдання для його проходження
г) Ознайомитися зі своїми результатами
д) Перехід до пазлів створених користувачем
е) Екран з уроком складатиметься із:
ж) Навчального матеріалу (текст, зображення)

з) Можливості переходу до завдання на основі поточного уроку

и) Можливості повернутися на головний екран

к) Можливості перейти на офіційну документацію по матеріалам із поточного навчального матеріалу

Екран з уроком матиме у будові:

а) Набір із мінімум 3, максимум 5 завдань

б) Секундомір який фіксує час, за який будуть пройдені завдання

в) Можливість повернутися на головну сторінку

г) Можливість перейти на навчальний матеріал на основі якого побудоване завдання

1.5 Висновок до розділу 1

З сучасним темпом розвитку технологій для людини важливо встигати у цій гонці, так як якщо вона не буде встигати, її потенційні можливості впадуть в рази. Для підтримки рівня знань та оцінки якості засвоєння нової інформації людьми використовується система тестувань. Існує багато різноманітних видів тестування, серед яких після дослідження найкориснішим можна виділити тести типу Drag drop які і було вирішено реалізувати. Для охоплення як можна більшої кількості потенційних користувачів було вирішено використовувати крос-платформні рішення у розробці програмного забезпечення. Одним із найкращих крос-платформних рішень стало використання у розробці Corona SDK який працює у сукупності із мовою програмування Lua, що дозволяє отримати неймовірний баланс у швидкодії коду та простоти його написання.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ПРОЕКТУ

2.1 Архітектури додатку

При розробці програмного забезпечення важливо не лише те, щоб програма добре працювала, але і щоб вона була добре організована. За це відповідає архітектура програми. Тож якщо брати до уваги, що складність програми зазвичай збільшується в кілька разів швидше ніж розміри самої програми, то відсутність заздалегідь продуманої архітектури загрожує швидкою втратою контролю над програмою та процесом її розробки.

Архітектура програмного забезпечення - сукупність найважливіших рішень про організацію програмної системи. Архітектура включає:

а) вибір структурних елементів та їх інтерфейсів, за допомогою яких складена система, а також їх поведінки в рамках співпраці структурних елементів;

б) з'єднання обраних елементів структури і поведінки у все більш крупні системи;

в) архітектурний стиль, який спрямовує всю організацію – всіх елементів, їх інтерфейсів, їх співпрацю та їх з'єднання.

При цьому загальноприйнятого визначення «архітектури програмного забезпечення» не існує. Так, сайт Software Engineering Institute наводить більше ніж 150 визначень цього поняття.

Архітектура відіграє велику роль на початковому етапі тому що

програму з хорошою архітектурою простіше розуміти, редагувати, тестувати та розширювати у майбутньому. Можна виділити список досить універсальних критеріїв для хорошої архітектури: ефективність системи, гнучкість системи, можливість розширення системи, масштабування процесу розробки, тестування системи, повторне використання та простий супровід.

При розробці архітектури додатку «PuzzLearn», до уваги було взято три головних аспекти: залучення до проекту бази даних для спрощення з роботою інформацією, використання нативних інструментів операційної системи, та активне використання вбудованих у Corona SDK бібліотек та плагінів, для забезпечення більшої гнучкості (Рис. 2.1. Архітектура додатку).

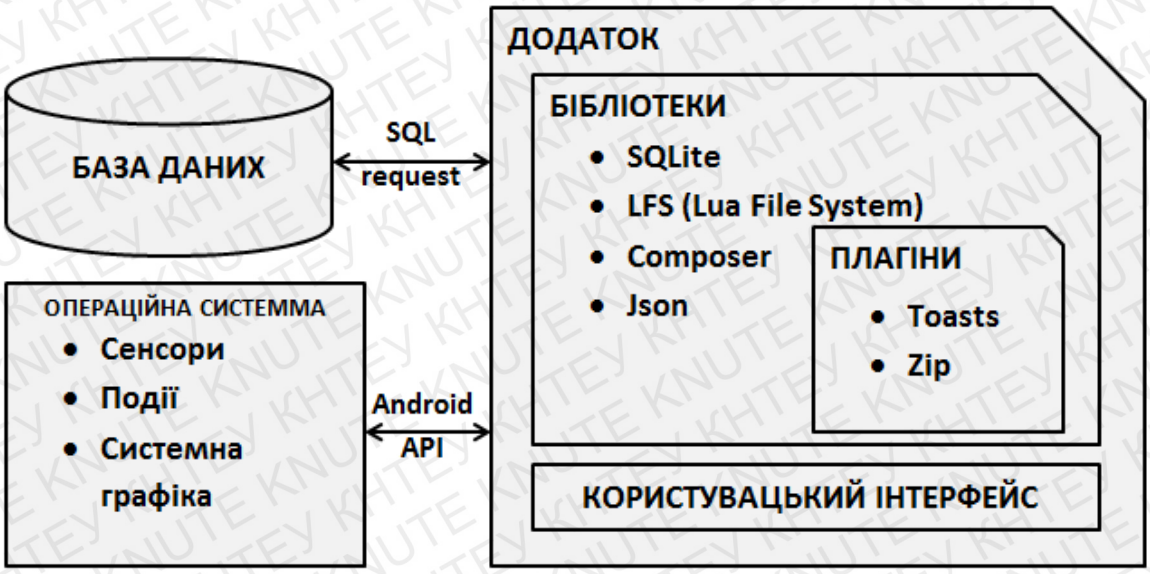

Рис. 2.1. Архітектура додатку

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. Більш по науковому це звучить як декомпозиція. 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.

2.2 Систематизація файлів проекту

При розробці будь якого програмного забезпечення при умові якщо відсутня систематизація фалів, що використовуються в проекті, то настає хаос, на розгрібання якого може піти великий проміжок часу, особливо коли вже написано багато коду де ідуть посилання на файли. Окремої розмови заслуговують файли що попросту можуть загубитися у папці проекту які вже і не потрібні, але через неналежне оформлення робочого простору ці файли залишаються, потрапляють у фінальну зборку програми де можуть не лише займати зайве місце, що явно не є плюсом для фінального користувача, а ще і можуть у самий не зручний час зламати додаток. Саме через це, систематизація файлів досить важливий аспект у розробці, якому варто надати належну увагу вже при створенні проекту.

Corona SDK як і будь який інший сучасний інструмент має досить просту роботу з файловим менеджером, коли у конкурентів існує вбудоване у редактор вікно з роботою файлами, у Corona цього попросту немає, тож весь менеджмент лягає цілком на розробника і сторонні програми. Лише при створенні пустого проекту Corona створює дванадцять елементів, два із яких є папками в яких зберігаються інші

файли

Для зручності роботи з власними файлами буде створена папка res у якій і будуть знаходитися усі файли призначені для розробки. Додаток PuzzLearn матиме майже всі типи файлів, тож, для зручного управління ними для кожного типу будуть створенні власні папки:

а) images для зберігання усього графічного матеріалу;

б) font що буде містити файли типу .ttf призначені для відображення тексту;

в) sound для зберігання не значної кількості аудіо файлів;

г) files у конкретно данному прикладі папка призначена для файлів формату бази даних;

д) scens найголовніша папка яка буде зберігати файли формату .lua які будуть містити у собі код програми;

Папка scens важлива тому-що як було сказано писати весь код в одному файлі поганий тон, саме тому при розробці додатку у Corona використовується бібліотека під назвою Composer. Composer являється офіційною системою створення та управління сценами у Corona. Сцена – це просто файл формату .lua які містять у собі код та зберігається як окремий файл. По своїй суті в програмному плані сцена схожа з аналогом у Android Studio де для розподілення коду використовують активності. Звичайно використання бібліотеки створює додатковий код, але це досить низька ціна за комфортне управління проектом. У даному проекті кількість сцен коливатиметься у кількості одинадцяти штук, основними з яких будуть: menu, urok, pazla, darkSide, makePazzle та myPazzle. Таким чином фінальна версія файлів матиме структуру зображену на (Рис. 2.2. Дерево файлів у папці res)

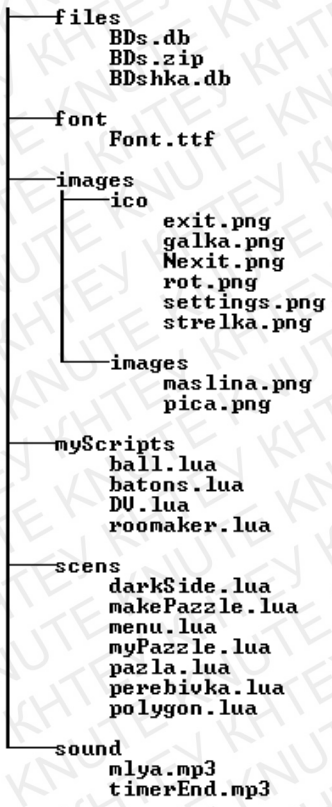

Рис. 2.2. Дерево файлів у папці res

2.3 Проектування користувацького інтерфейсу

Одним із найважливіших критеріїв при виборі того чи іншого програмного забезпечення для користувачів все частіше виступає зовнішній вигляд додатку, його оформлення, стилістика та інше. Інтерфейс та його зовнішній вигляд грає велику роль у майбутній популярності додатку, так як саме від зовнішнього вигляду складається перше враження у користувача про певний програмний продукт, таким чином навіть найкраща по функціональності та логіці програма з погано розробленим інтерфейсом може програти у конкуренту з куди більш слабкою в технічному плані реалізацією, але при цьому краще виконаним інтерфейсом. Для створення таких якісних користувацьких інтерфейсів у команді розробників обов'язково повинен бути залучений дизайнер-проектувальник. Такий професіонал здатний вирішити багато

проблем так як саме він стандартизує зовнішній вигляд додатку: його кнопки, іконки, текстові данні та інші елементи інтерфейсу, створюючи єдину екосистему в якій користувачу буде приємно проводити час виконуючи поставлені для додатку задачі

При проектуванні інтерфейсу додатку «PuzzLearn» головним прикладом для наслідування виступав розроблений компанією Google стиль інтерфейсу Material design. Його особливість полягає у відображені всіх елементів додатку, як речей з реального світу, такі як папір, чорнила та найголовніше тіні. Завдяки офіційному середовищу для розробки Android Studio проектування інтерфейсу займає значно менше часу ніж створення його власноруч з нуля, проте існує обмеження по операційним платформам, що збільшує час адаптації додатку під інші операційні системи.

Головний функціонал додатку складатиметься із трьох основних екранів:, екрану з навчальним матеріалом, екраном з тестовим матеріалом та головним екраном, який в свою чергу складається із трьох під екранів.(Рис. 2.3. Три під екрани головного екрану).

Рис. 2.3. Три під екрани головного екрану

Головний екран складається с назви проекту, карточок, списків тестів і навчальних матеріалів та навігаційного елементу, що дає доступ до вкладок Home (Рис. 2.4. будова вкладки Home), Learn, Tests.

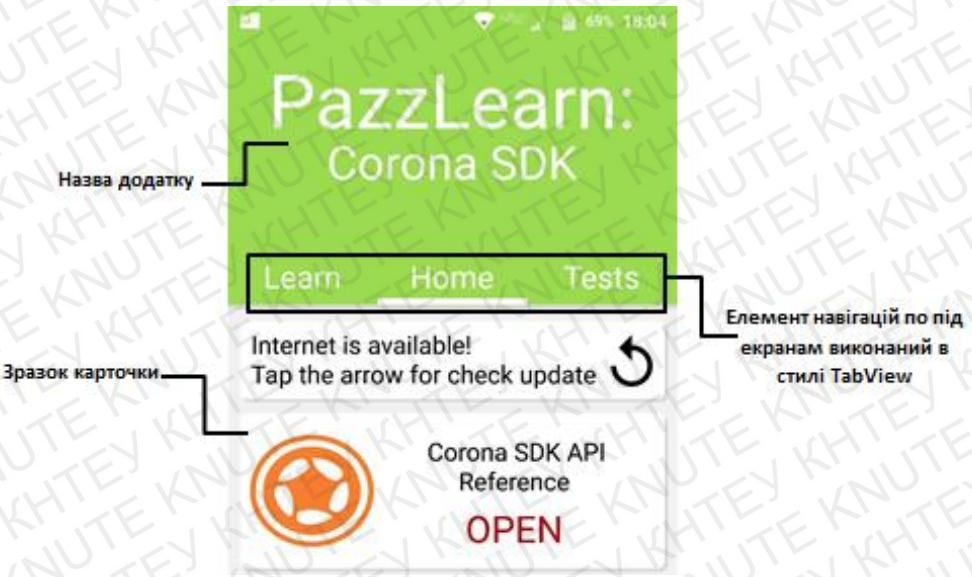

Рис. 2.4. будова вкладки Home

Проте найважливішою частиною додатку є екрани з навчальним матеріалом та тестами до нього, якщо з навчальним матеріалом все просто, то з тестами структура дещо складніша так як варто проробити її зручною як з плану розробки так і взаємодії з користувачем. Основними елементами тестів будуть чотири речі: завдання, місце для генерування шматочків пазлів, місце для складання тесту та власне самих тестів (Рис 2.5. Будова екрану з тестами).

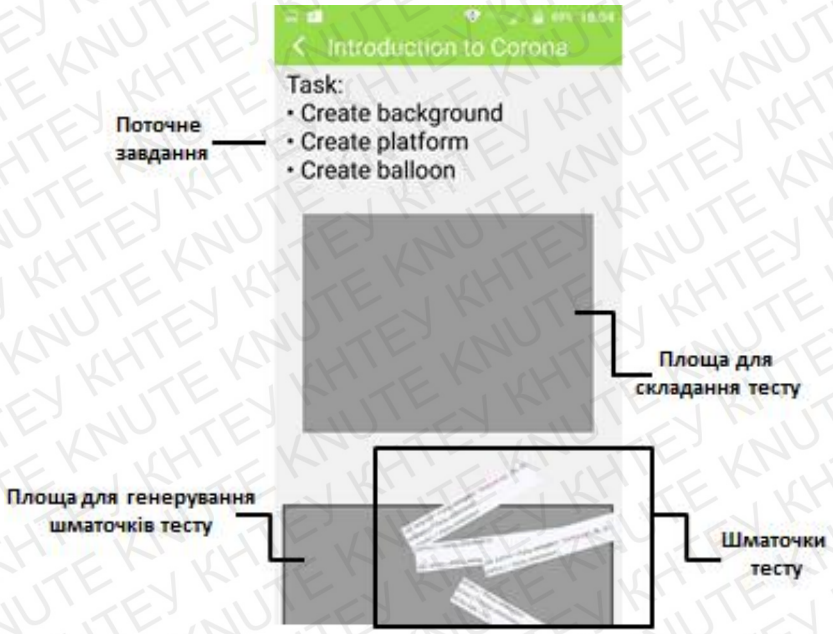

Рис.2.5. Будова екрану з тестами

Попри основних елементів також існують додаткові: кнопка назад, оформлення заголовку та кнопка переходу до наступного тесту. Також можливе винесення списку із завданням до окремої викликаємої користувачем області.

2.4 Висновок до розділу 2

У другому розділі розглянута та спроектована архітектура додатку на основі технічного завдання, що дозволить більш продуктивно налагодити подальший процес розробки додатку. Була обрана стилістика додатку та спроектований користувацький інтерфейс дотримуючись правил оформлення Material design. Та був налагоджений робочий простір перед початком безпосередньої розробки додатку.

.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Побудова інтерфейсу

Вже як дев’ять років Corona SDK активно розвивається розширюючи свої можливості та надаючи розробникам новий функціонал так у 2013 році публіці була представлена бібліотека widget 2.0. Ця бібліотека включає у себе готові рішення таких популярних конструкцій як: button, picker wheel, progress view, segmented control, slider, spinner, stepper, switch, tab bar, table view. Всі ці інструменти прості та зручні у використанні, але список можливих налаштувань досить скудний, саме тому, навіть кнопки доводиться реалізовувати самому з нуля. Все складне складається з простого, так і у побудові інтерфейсу у Corona, будь який складний інтерфейс складається з таких простих елементів як прямокутники різних форм. Такий підхід можна назвати як позитивним так і негативним, з однієї сторони він забезпечує неймовірну гнучкість налаштування зовнішнього виду додатку, проте з іншого це може значно збільшити час розробки при умові розробки складного інтерфейсу. Також до спірного моменту можна віднести відсутність графічного конструктора інтерфейсів, що обумовлюється кросплатформністю Corona, проте для когось, це може бути зайвою причиною, щоб обрати саме Corona SDK у своїй розробці.

Розглянемо побудову інтерфейсу головної сцени menu.lua. Для самого початку варто змінити стандартний фон та змінити зовнішній вигляд статус бару. За зміну зовнішнього вигляду статус бару відповідає

команда display.setStatusBar(тип), всього існує шість типів статус бару: hidden, default, translucent, dark, lightTransparent, darkTransparent. У випадку з світлим тоном сцени найкраще підійде варіант з світлим стилем та прозорим фоном: LightTransparentStatusBar. Після цього змінюється задній фон за допомоги команди: display.setDefault("background", 0.95, 0.95, 0.95). Це робиться для того, щоб майбутні білі картки інтерфейсу контрастували створюючи ефект «цифрового паперу». Кожен інтерфейс будується с простих елементів, у даному випадку це прямокутник. Викликається прямокутник за допомоги простої команди display.newRect(властивості). Для відображення у властивості можна передавати таблицю з повним спектром властивостей створену раніше, або вказувати стандартні напряму: групу до якої належить об'єкт, позицію по осі X, позицію по осі Y, ширину та висоту. Але при прямому вказуванні можна вводити не всі бажані параметри, тож для прямокутника після його створення варто окремо передати його колір заливки завдяки rectName:setFillColor(r,g,b,a). Тож за допомоги цих команд можна зібрати код який створить зелений прямокутник у верхній частині дисплею (Рис. 3.1. Код реалізації прямокутника зеленого кольору).

```
topBox = display.newRect(topGroup, C_X,_Y/4/2, _X, _Y/2 )
topBox:setFillColor(0.61, 0.86, 0.32)
```

Рис. 3.1. Код реалізації прямокутника зеленого кольору

Після цього для більшої схожості з Material design прямокутнику варто додати тінь, тінь складається з такого ж прямокутника лише опущеним трохи нижче та додати йому показник альфи що дорівнюватиме 0.1.

Текст майже повністю повторює логіку прямокутника, єдина відмінність це додатково вказувати шрифт який використовується, та

розмір вказується однією змінною. Проте щоб згодом зробити милу анімацію при змінні розділу, варто розбити текст на декілька частин помістивши їх у два ряди.

Останнім і найголовнішим пунктом у формуванні першого повноцінного елементу сцени є створення під назвою додатку інструменту tabview. Для його створення будуть використані чотири прямокутника: три для кнопок та один для підкреслення обраного пункту. Поверх кнопок будуть розміщенні елементи типу текст з підписами розділів. На кожну таку кнопку буде витрачатись третина по ширині від ширини самого дисплею та висота у 150 віртуальних пікселів. Таким чином використовуючи даний код (Рис. 3.2. Код реалізації елементу TabView) у фінальному результаті вийде готова верхня частина додатку

```
learnTextFon = display.newRect(topGroup,C_X/3,870,_X/3,150)
learnTextFon:setFillColor(0.61, 0.86, 0.32)
learnText = display.newText(topGroup, "Learn", C_X/3, learnTextFon.y, font, 100)

homeTextFon = display.newRect(topGroup,C_X,870,_X/3,150)
homeTextFon:setFillColor(0.61, 0.86, 0.32)
homeText = display.newText(topGroup, "Home", C_X, 870, font, 100)

testTextFon = display.newRect(topGroup,_X- C_X/3,870,_X/3,150)
testTextFon:setFillColor(0.61, 0.86, 0.32)
testText = display.newText(topGroup, "Tests", _X- C_X/3, 870, font, 100)

tabLine = display.newRect(topGroup, C_X, 950, _X/3,20)
```

Рис. 3.2. Код реалізації елементу TabView

За анімації у Corona SDK відповідають два інструменти: класичний transition та відносно новий animation, принцип їх роботи відрізняється мінімально, тож очевидного фаворита немає. Робота з transition заключається у визові функції, передачі об’єкту для маніпуляцій, та властивості які варто застосувати до об’єкту. Наприклад код «transition.to(container, { height = containerY, width = containerX, transition = easing.inOutQuint, onComplete = newFunc })» задає об’єкту

container нові властивості висоти та ширини з згладжуванням анімації типу easing.inOutQuint, після завершення якої викликається функція newFunc. Такий простий інструмент як transition дає великі можливості у питані анімацій в додатку, с використанням котрої додаток все більше відповідає нормам Material design. Саме transition забезпечує плавну анімацію при переходах між вкладками у головному меню.

3.2 Створення логіки додатку

Як і в будь-якій програмі все починається з main. При використанні composer файл main.lua варто проінформувати про використання даної бібліотеки, досягається це завдяки підключенню через функцію require: local composer = require ("composer"). Після підключення, яке необхідне у кожній сцені, відкривається можливість переходу на будь-яку іншу сцену, але перед тим рекомендовано ініціювати при потребі так зване зерно для рандомайзеру, зазвичай у якості зерна використовують поточний час, тож у якості параметру передається час пристрою, виглядатиме такий код так: math.randomseed(os.time()). Також, при потребі існує можливість вимкнути таймер гасіння дисплею, що є зручним при тестуванні, але не бажане у використанні на постійній основі. Після завершення базових налаштувань відбувається сам перехід на сцену, у цьому випадку до сцени головного меню, досягти цього можливо завдяки функції composer.gotoScene(). При використанні ком позера для кожної сцени існує певне додаткове структурне налаштування, які варто дотримуватися щоб composer міг розглядати файли формату .lua саме як сцени. Налаштування включають в себе дві строки для ініціювання сцени, чотири функції слухачів для обробки подій, що генерує composer, чотири строки коду для ініціювання цих функцій прослуховувача та основну інформацію для

створення зв’язку сцени з composer. Сцена має чотири різних стану життєвого циклу:

а) create. Відбувається, коли сцена вперше створюється але ще не відображається;

б) show. Відбувається перед та одразу після появи сцени на екрані;

в) hide. Відбувається перед та одразу після зникнення сцени з екрану;

г) destroy. Відбувається при знищенні сцени;

При такій структурі файлу також змінюється і логіка побудови додатку наприклад оголошувати змінні варто до створення сцени, а ініціювати змінні вже у функції створення сцени, так можна робити і з функціями, проте у разі розробки гри, запускати ігровий цикл або фізику варто тільки після відображення сцени. Після ініціалізації всього графічного інтерфейсу у справу вступає функціональна частина коду. Першим і самим головним у додатку PuzzLearn є відображення списку навчальних матеріалів та завдань які зберігаються у файлі типу бази даних, тож перед роботою з такими файлами варто підключити бібліотеки для роботи з sqlite3. Підключення бібліотеки аналогічне підключенню composer. Перша версія бази даних зберігатиметься у самому додатку, тож для її виклику для неї треба вказати шлях: local path = system.pathForFile("BDs.db"), після цього відкривається доступ до самої бази даних за допомоги sqlite3.open(path). Як тільки шлях ініціалізовано, а файл відкритий, можна приступити до заповнення tableView. Робиться це завдяки циклу for (Рис. 3.3. Цикл заповнення елементу TableView)

```
for row in db:nrows("SELECT * FROM lernTable") do

    --
    local rowParams=
    {
        ID = row.id,
        Name = row.name,
        Check = row.completed,
    }

    scrollViewLearn:insertRow(
    {
        rowHeight = 150,
        params = rowParams,
        rowStrokeWidth = 20,
        rowStrokeColor = {0.95,0.95, 0.95},
    }
    )
end
```

Рис. 3.3. Цикл заповнення елементу TableView

Цикл відправляє запит на отримання таких змінних як ID, Name та Check з таблиці lernTable допоки вона не закінчиться, потім передаючи всі ці змінні у вигляді таблиці функції, яка відображає отримані результати у списку.

При взаємодії користувача із списком спрацьовує відповідний слухач (Рис. 3.4. Функція слухача списку з пазлами), який, в залежності від дії, викликає відповідну функцію.

```
onRowTouch2 = function(event)
    if event.phase == "tap" then
        toast.show(event.target.text .. " was chosen")
        local myParams =
        {
            papka = event.target.text,
            stage = 1
        }
        composer.gotoScene("scens.polygon",{effect = "slideUp", time = 1000, params = myParams} )
    elseif event.phase == "press" then
        toast.show(event.target.text .. " pressed")
    elseif event.phase == "release" then
    elseif event.phase == "swipeRight" then
        gotoHome()
    elseif event.phase == "swipeLeft" then
    end
end
```

Рис. 3.4. Функція слухача списку з пазлами

У випадку взаємодії зі списком пазлів, коли користувач натискає на елемент списку викликається функція переходу на нову сцену, передаючи у таблиці параметрів назву елементу, на який натиснув

користувач, це робиться для того, щоб у подальшому перейти у відповідну папку, в якій містяться елементи для проходження пазлу.

Завантаження шматочків пазлу відбувається шляхом простого перебору папки з назвою, яку отримали від попередньої сцени, в якій містяться пронумеровані файли формату .png, що після завантаження для більшого сприйняття за допомоги рандому розташовуватимуться на прямокутнику у зменшеній формі та під рандомним кутом в діапазоні від мінус тридцяти до плюс тридцяти градусів. За це відповідає функція movePiecesToTray (Рис. 3.5. Функція movePiecesToTray).

```
movePiecesToTray = function()
    local yStart = piecesTray.y - piecesTray.contentHeight/2 + 55
    local yEnd   = piecesTray.y + piecesTray.contentHeight/2 - 40
    local curY = yStart
    for i = 1, #puzzlePieces do
        local aPiece = puzzlePieces[i]
        aPiece.x = piecesTray.x + math.random( -400, 400 )
        aPiece.y = curY + math.random(-20,50)
        aPiece.inTrayX = aPiece.x
        aPiece.inTrayY = aPiece.y
        aPiece.xScale = 0.5
        aPiece.yScale = 0.5
        aPiece.rotation = math.random( -30, 30 )
        aPiece.inTrayAngle = aPiece.rotation
        curY = curY + math.random( 60, 100)
        if( curY > yEnd ) then
            curY = yStart
        end

    end
end
```

Рис. 3.5. Функція movePiecesToTray

Після цього очікується взаємодія користувача із пазлами, за що відповідає найбільша та найважливіша функція onPuzzlePieceTouch. Функція складається із багатьох перевірок, перша перевірка заключається у перевірці елементу на який натискає користувач чи він ще активний, у випадку, якщо елемент вже знаходиться на своєму місці, він вважається не активним і функція повертає значення true припиняючи функцію. Якщо об’єкт активний йде перевірка стадії взаємодії: Began, Moved, Ended. При стані began об’єкт, на який

натиснули вирівнюється та повертається до звичайного розміру задаванням параметрів rotation та xScale, yScale відповідно. Після цього об’єкт переміщується поверх усіх об’єктів і йому задається фокус. Якщо стан дорівнює moved тоді викликається функція centerDrag, яка відповідає за переміщення об’єкту залежно від знаходження пальцю на дисплеї. І останній стан це ended після якого знімається фокус та йде перевірка чи відповідає даний об’єкт його запланованій позиції та в якщо умова істина робить об’єкт не активним прив’язуючи до координати.

3.3 Висновки до розділу 3

У даному розділі були повністю розглянуті засоби реалізації розробленого інтерфейсу та логіки додатку, проаналізовані максимальні можливості середовища Corona SDK. Розроблений базовий комплекс навчального додатку з можливістю подальшої модифікації та розширення області навчання. Вибраний інструмент 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 дипломного проекту було розглянуто питання процесу навчання завдяки використанню сучасних технологій та проаналізовані багато можливих способів та можливих інструментів для створення програмних навчальних додатків для різноманітних цільових платформ. Завдяки вибору крос-платформного інструменту для розробки Corona SDK була розроблена версія додатку для мобільних пристроїв, яку згодом можна переробити із мінімальними змінами для десктопних рішень.

Було розроблено структуру мобільного додатку із застосуванням популярного на мобільній платформі дизайну Material Design, який з кожним роком поступово охоплює і стаціонарні платформи як Windows. У процесі навчання використано працюючий курс, що дає впевнений старт для початку розробки із застосуванням інструментів для розробки Corona SDK.

У подальшому розвитку додатку можливе додавання реклами для отримання прибутку, додавання інших мов інтерфейсу та побудова більш раціонального у використанні на комп'ютерах інтерфейсу. Розширення вибору мов програмування для навчання та введення системи досягнень, 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. 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.
4. А. Ю. Берко, О. М. Верес; Організація баз даних: практичний курс : Навч. посіб. для студ. Нац. ун-т "Львів. політехніка". – Л., 2003. – 149 с.
5. Коннолліі Т., Бегг К. Бази даних. Проектування, реалізація і супровід. Теорія і практика = Database Systems: A Practical Approach to Design, Implementation, and Management. – 3-е изд. – М. : Вільямс, 2003. – 1436 с. – ISBN 0-201-70857-4.

Інтернет ресурси:

1. Документація редактору коду Sublime Text [Електронний ресурс]. – [Режим доступу]: https://www.sublimetext.com/support
2. Офіційна документація Corona SDK. [Електронний ресурс] – Режим

доступу: https://docs.coronalabs.com/

ДОДАТОК А

ЛІСТИНГ КОДУ ДОДАТКУ

```
local widget = require( "widget" )
local composer = require( "composer" )
local toast = require('plugin.toast')
local DSize = require("DSize")
local socket = require("socket")
local json = require( "json" )
local widget = require("widget")
                require("sqlite3")
local path = system.pathForFile("BDs.db")
local function doesFileExist(theFile, path)
        local thePath = path or system.DocumentsDirectory
        local filePath = system.pathForFile(theFile, thePath)
        local results = false
        local file = io.open(filePath, "r")
        if file then
                io.close(file)
                results = true
        end
        return results
end

if doesFileExist("newDB.db", system.DocumentsDirectory) == true then
        path = system.pathForFile("newDB.db", system.DocumentsDirectory)
end
```

```
local haveUpdate = true
local path2 = system.pathForFile("saves.json", system.DocumentsDirectory)
local db = sqlite3.open(path)
local scene = composer.newScene()
local currScene = composer.getSceneName( "current" )
local PazzLearn = "PazzLearn"
local settingsB; local shadow; local topBox; local nameText; local tabLin; local
learnText
local topGroup; local menuBFon; local menuBText; local upDateIco; local
upDateText; local upDateRect
local shadow; local scrollView; local slideUpText;local Baton; local picaImg;
local slideUpText2
local scrollViewLearn; local scrollViewTests; local exitCheck=false;local
checkMenu = false
local exitFun; local exitTimer; local onKeyEvent; local seeMenu
local gotoDarkSide,gotoPazzle
local gotoLearn, gotoHome, gotoTests
local scrollListenerHOME, scrollListenerLEARN, scrollListenerTESTS

function scene:create( event )

    local sceneGroup = self.view

    display.setStatusBar( display.LightTransparentStatusBar)
    display.setDefault( "background", 0.95, 0.95, 0.95 )

    fillUpdateGroup = function()
```

```
	local test = socket.tcp()
	test:settimeout(1, 't')

	local testResult = test:connect("www.google.com", 80)

	if not(testResult == nil) then
		upDateTextAviable.isVisible = true
		upDateTextNoAviable.isVisible = false
		upDateIco.isVisible = true

	else
		upDateTextAviable.isVisible = false
		upDateTextNoAviable.isVisible = true
		upDateIco.isVisible = false

	end

	test:close()
	test = nil
end

onRowTouch = function(event)

	if event.phase == "tap" then

	elseif event.phase == "press" then
```

```
        elseif event.phase == "release" then

        elseif event.phase == "swipeRight" then
        elseif event.phase == "swipeLeft" then

        end
    end

    local function networkListener( event )
      if ( event.isError ) then
        toast.show( "Network error: ", event.response )
                native.setActivityIndicator(false)
      elseif ( event.phase == "began" ) then
        if ( event.bytesEstimated <= 0 ) then
          toast.show( "Download starting, size unknown" )
        else
          toast.show( "Download starting, estimated size: " ..
event.bytesEstimated )
        end

      elseif ( event.phase == "progress" ) then
        if ( event.bytesEstimated <= 0 ) then
          toast.show( "Download progress: " .. event.bytesTransferred )
        else
          toast.show( "Download progress: " .. event.bytesTransferred .. " of
estimated: " .. event.bytesEstimated )
        end
```

```
        elseif ( event.phase == "ended" ) then
            toast.show( "Download complete, total bytes transferred: " ..
event.bytesTransferred )
            native.setActivityIndicator(false)
            scrollViewLearn:deleteAllRows()


                path = system.pathForFile("newDB.db",
system.DocumentsDirectory)
                local db = sqlite3.open(path)


    for row in db:nrows("SELECT * FROM lernTable") do

                --
                local rowParams=
                {
                    ID = row.id,
                    Name = row.name,
                    Check = row.completed,
                }

                scrollViewLearn:insertRow(
                {
                    rowHeight = 150,
                    params = rowParams,
```

```
                rowStrokeWidth = 20,
                rowStrokeColor = {0.95,0.95, 0.95},
            }
            )
            end
    end
    end

    local function downloadBD()
        local params = {}

        params.progress = "download"
        params.response =
        {
            filename = "newDB.db",
            baseDirectory = system.DocumentsDirectory
        }

        network.request(
"https://drive.google.com/uc?export=download&id=1CN_ErNUgWoVIa8uV63
cQ4Ct5_1dwLjr2",
        "GET", networkListener,  params )

    end

    local function updateMask( event )
      if ( event.action == "clicked" ) then
```

```
        local i = event.index
        if ( i == 1 ) then

        elseif ( i == 2 ) then

                local test = socket.tcp()
                test:settimeout(1, 't')
                local testResult = test:connect("www.google.com", 80)

                if not(testResult == nil) then
                        native.setActivityIndicator(true)
            downloadBD()
                else
                        toast.show("COMRAD, NO INTERNET
CONNECTION!")
                end

            test:close()
            test = nil

        end
    end
end

local function checkUpdate( )

    local test = socket.tcp()
```

```
        test:settimeout(1, 't')

        local testResult = test:connect("www.google.com", 80)

        if not(testResult == nil) then

          toast.show("I HAVE INET")
          transition.to( upDateIco, {rotation = upDateIco.rotation-360,
time=1000, transition=easing.inOutQuint} )
            timer.performWithDelay( 1000, function()

            if haveUpdate == true then

                local alert = native.showAlert("We have update", "I
found update of DB. Wanna to download and update?", {"Not
now","Dowlod"'},updateMask)
                haveUpdate = false
                toast.show("LAST VERSION OF DB WAS INSTAL")
                end
            end
            else
                print("Internet access is not available")
            toast.show("COMRAD, NO INTERNET CONNECTION!")
            end

        test:close()
        test = nil
```

```
end

exitFun = function()
	if exitCheck == true then
		native.requestExit()
	else
		toast.show('Press back again for exit')
	end

end

exitTimer = function()
	exitCheck=false
end

onKeyEvent = function( event )

	if (event.keyName == "back") then
		if (event.phase=="down") then
			exitFun()
			timer.performWithDelay( 2000, exitTimer)

			exitCheck=true

		return true
		end
	return false
```

```
        end
    end

    seeMenu = function()

        if checkMenu == false then
            transition.to(menuGroup, {time=100,x=settingsB.x-200,
alpha=1})
            checkMenu = true

        elseif checkMenu == true then
            transition.to(menuGroup, {time=100,x=settingsB.x+350,
alpha=0})
            checkMenu = false
        end
    end

    gotoDarkSide = function()
        composer.gotoScene("scens.darkSide","flipFadeOutIn", 200)
    end

    gotoPazzle = function()
        composer.gotoScene("scens.pazla","slideUp", 1000)
    end

    gotoPolygon = function()
        composer.gotoScene("scens.polygon","slideUp",1000)
```

```
end

gotoLearn = function()
	transition.to( tabLine, { time=200, x=learnText.x } )
	transition.to( topGroup, { time=200, y=-560, } )
	transition.to( mainGroup, { time=200, x=_X, y=-560, } )
	transition.to( settingsB, { time=200, alpha=1 } )
	transition.to(nameText, {time=200, x=290 , y=730 , size=100 })
	transition.to(nameText2, {time=200, x=820 , y=730 , size=100})
end

gotoHome = function()
	transition.to( tabLine, { time=200, x=homeText.x } )
	transition.to( topGroup, { time=200, y=0 } )
	transition.to( mainGroup, { time=200, x=0, y=0 } )
	transition.to( smallLogo, { time=200, alpha=0 } )
	transition.to( settingsB, { time=200, alpha=0 } )
	transition.to(nameText, {time=200, x=C_X,y=300 , size=240 })
	transition.to(nameText2, {time=200, x=C_X , y=500 , size=150})
end

gotoTests = function()
	transition.to( tabLine, { time=200, x=testText.x } )
	transition.to( topGroup, { time=200, y=-560 } )
	transition.to( mainGroup, { time=200, x=-_X, y=-560 } )
	transition.to( settingsB, { time=200, alpha=1 } )
	transition.to(nameText, {time=200, x=290 , y=730 , size=100 })
```

```
    transition.to(nameText2, {time=200, x=820 , y=730 , size=100})
end

scrollListenerHOME = function( event )
    local phase = event.phase
    if ( phase == "began" ) then print( "Scroll view was touched" )
    elseif ( phase == "moved" ) then print( "Scroll view was moved" )
    elseif ( phase == "ended" ) then print( "Scroll view was released" )
    end
    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" )
            gotoTests()
        elseif ( event.direction == "right" )  then
            gotoLearn()
        end
    end
    return true
end


scrollListenerLEARN = function( event )
    local phase = event.phase
```

```
        if ( phase == "began" ) then print( "Scroll view was touched" )
        elseif ( phase == "moved" ) then print( "Scroll view was moved" )
        elseif ( phase == "ended" ) then print( "Scroll view was released" )
        end
        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" )
                        gotoHome()
                elseif ( event.direction == "right" )  then
                        print("uhu")
                end
        end
        return true
end

scrollListenerTESTS = function( event )
        local phase = event.phase
        if ( phase == "began" ) then print( "Scroll view was touched" )
        elseif ( phase == "moved" ) then print( "Scroll view was moved" )
        elseif ( phase == "ended" ) then print( "Scroll view was released" )
        end
        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("uhu")
            elseif ( event.direction == "right" )  then
                gotoHome()
            end
        end
        return true
    end

    onRowTouch = function(event)

        if event.phase == "tap" then
            toast.show(event.target.text .. " was chosen")
            local myParams =
            {
                papka = event.target.text
            }
            composer.gotoScene("scens.pazlala",{effect = "slideRight",
time = 1000, params = myParams} )
        elseif event.phase == "press" then
            toast.show(event.target.text .. " pressed")
        elseif event.phase == "release" then
                elseif event.phase == "swipeRight" then
        elseif event.phase == "swipeLeft" then
```

```
                gotoHome()
            end
        end

        onRowTouch2 = function(event)

            if event.phase == "tap" then
                toast.show(event.target.text .. " was chosen")
                local myParams =
                {
                    papka = event.target.text,
                    stage = 1
                }
                composer.gotoScene("scens.polygon",{effect = "slideUp",
time = 1000, params = myParams} )
            elseif event.phase == "press" then -- реакция на нажатие
                toast.show(event.target.text .. " pressed")
            elseif event.phase == "release" then -- реакция на отжим
            elseif event.phase == "swipeRight" then
                gotoHome()
            elseif event.phase == "swipeLeft" then
            end
        end

        scrollView = widget.newScrollView(
    {
            mainGroup,
```

```
    top = 960,
    width = _X,
    height = 1600,
    scrollWidth = 0,
    scrollHeight = 0,
            horizontalScrollDisabled = false,
    listener = scrollListenerHOME,
            bottomPadding = -1025,
            rightPadding = -240,
            backgroundColor = { 0.95, 0.95, 0.95 }

})

  local function onRowRender(event)
            local row = event.row
            local fontSize = 100
            local rowHeight = row.height/2

            local options_id =
            {
                    parent = row,
                    text = row.params.ID .. ":",
                    x = 150,
                    y = rowHeight,
                    font = font,
                    fontSize = fontSize,
            }
```

```
        row.id = display.newText(options_id )
        row.id.anchorX = 0
        row.id:setFillColor(0,0,0)
        row.id.anchorX = 1

        local options_name =
        {
            parent = row,
            text = row.params.Name,
            x = row.id.x+25,
            y = rowHeight,
            font = font,
            fontSize = fontSize,
        }
        row.text = row.params.Name
        row.name = display.newText(options_name)
        row.name:setFillColor(0,0,0)
        row.name.anchorX=0

        local checker = row.params.Check
        if checker == 1 then
            row.check = display.newImage(row, "ico/galka.png", _X -
75, rowHeight)
        end
    end
```

```
    scrollViewTests = widget.newTableView(
{
        mainGroup,
    top = 1000,
    left = _X,
    width = _X,
    height = 2160,
    scrollWidth = 0,
    scrollHeight = 0,
        horizontalScrollDisabled = false,
    listener = scrollListenerTESTS,
        status = tests,
        onRowRender = onRowRender,
    onRowTouch = onRowTouch2,
        hideBackground = true,
     })

scrollViewLearn = widget.newTableView(
 {
        mainGroup,
    top = 960,
    left = -_X,
    width = _X,
    height = 2160,
    scrollWidth = 0,
    scrollHeight = 0,
        horizontalScrollDisabled =false,
```

```
        friction = 0.01,
    listener = scrollListenerLEARN,
    onRowRender = onRowRender,
    onRowTouch = onRowTouch,
    hideBackground = true

})

    for row in db:nrows("SELECT * FROM lernTable") do

        local rowParams=
        {
            ID = row.id,
            Name = row.name,
            Check = row.completed,
        }

        scrollViewLearn:insertRow(
        {
            rowHeight = 150,
            params = rowParams,
            rowStrokeWidth = 20,
            rowStrokeColor = {0.95,0.95, 0.95},
        })

        scrollViewTests:insertRow(
        {
```

```
                rowHeight = 150,
                params = rowParams,
                rowStrokeWidth = 20,
                rowStrokeColor = {0.95,0.95, 0.95},
            })
        end

        slideUpText = display.newText("Листай вверх", C_X, _Y + 500,font,
150)
        slideUpText:setFillColor(0,0,0)
        scrollView:insert( slideUpText )


        picaImg = display.newImageRect( "pica.png", _X, 1024 )
        picaImg.x=C_X
        picaImg.y=slideUpText.x+1500
        scrollView:insert( picaImg )

        slideUpText2 = display.newText("Ты долистался приятель",
C_X,picaImg.y+200,font, 100)
        slideUpText2:setFillColor(1,1,0)
        scrollView:insert( slideUpText2 )

        local upDateGroup = display.newGroup()
        scrollView:insert(upDateGroup)
```

```
	upDateRect = display.newRoundedRect(upDateGroup, C_X, 180, _X-80, 250, 20)
	local upDateRectShadow = display.newRoundedRect(upDateGroup, upDateRect.x+10, upDateRect.y+10, _X-80, 250, 20)
	upDateRectShadow:setFillColor(0,0,0,0.1)
	upDateRect:toFront()

	upDateTextAviable = display. newText(upDateGroup, "Internet is available! \nTap the arrow for check update", 75 , upDateRect.y+10 , upDateRect.width-250, upDateRect.height-20, font,  80)
		upDateTextAviable:setFillColor(0,0,0)
		upDateTextAviable.anchorX = 0
		upDateTextAviable.isVisible = false
	upDateTextNoAviable = display. newText(upDateGroup, "No Internet is available. \nTurn on internet for check update.", 75 , upDateRect.y+10 , upDateRect.width-40, upDateRect.height-20, font,  80)
		upDateTextNoAviable:setFillColor(0,0,0)
		upDateTextNoAviable.anchorX = 0
		upDateTextNoAviable.isVisible = false

	upDateIco = display.newRect(upDateGroup, upDateRect.x+575, 200 , 200, 200)
	upDateIco.anchorY=0.55
	upDateIco.fill = { type = "image", filename = "ico/strelka.png" }

	local coronaLinkGroup = display.newGroup()
		scrollView:insert(coronaLinkGroup)
```

```
	coronaLinkRect = display.newRoundedRect(coronaLinkGroup, C_X,
350, _X-80, 500, 20)
		local coronaLinkRectShadow =
display.newRoundedRect(coronaLinkGroup, coronaLinkRect.x+10,
coronaLinkRect.y+10, coronaLinkRect.width, coronaLinkRect.height, 20)
		coronaLinkRectShadow.anchorY = 0
		coronaLinkRectShadow:setFillColor(0,0,0,0.1)
		coronaLinkRect:toFront()
		coronaLinkRect.anchorY = 0

	coronaLinkLogo = display.newImageRect( coronaLinkGroup,
"images/corona.png", 557, 406)
	coronaLinkLogo.x, coronaLinkLogo.y = coronaLinkRect.x-450,
coronaLinkRect.y+coronaLinkRect.height/2
	coronaLinkText = display.newText(coronaLinkGroup, " Corona SDK API
\n	Reference", coronaLinkRect.x - 100 , coronaLinkRect.y+170 , font,  80,
"center")
		coronaLinkText:setFillColor(0,0,0)
		coronaLinkText.anchorX = 0

	coronaLinkLink = display.newText ( coronaLinkGroup, "OPEN",
coronaLinkText.x+300, coronaLinkText.y + 200, font, 120)
	coronaLinkLink:setFillColor(0.7,0,0)
	coronaLinkLink:addEventListener( "tap", function()
		system.openURL( "https://docs.coronalabs.com/api/index.html" )
	end )
```

```
	Baton = display.newText(scrollViewTests, "pazleblin", C_X, _Y, font,
120)
		Baton:setFillColor(0,0,0)
		scrollView:insert(Baton)

	Batonka = display.newText(scrollViewTests, "polygon", C_X, _Y+200,
font, 120)
		Batonka:setFillColor(0,0,0)
		scrollView:insert(Batonka)

	mainGroup = display.newGroup()
	sceneGroup:insert (mainGroup)
	topGroup = display.newGroup()
	menuGroup = display.newGroup()

	sceneGroup:insert (topGroup)

	mainGroup:insert (scrollView)
	mainGroup:insert (scrollViewLearn)
	mainGroup:insert (scrollViewTests)
	sceneGroup:insert (menuGroup)
	menuGroup.x = mainGroup.x+_X+250
	menuGroup.alpha= 0

	shadow = display.newRect(topGroup, C_X,_Y/4/2+15, _X, _Y/2)
	shadow:setFillColor(0,0,0,0.1)
```

```
	topBox = display.newRect(topGroup, C_X,_Y/4/2, _X, _Y/2 )
	topBox:setFillColor(0.61, 0.86, 0.32)

	nameText = display.newText(topGroup, PazzLearn .. ":", C_X,300,font,
240)
	nameText2 = display.newText(topGroup, "Corona SDK", C_X,500,font,
150)

	smallLogo = display.newText(topGroup, PazzLearn.. ": Corona SDK"
,30, 730, font, 100 )
	smallLogo.alpha=0
	smallLogo.anchorX=0

	settingsB = display.newImageRect(topGroup, "ico/settings.png", 100,100)
	settingsB.x, settingsB.y = _X-100, smallLogo.y
	settingsB.fill.effect = "filter.invert"
	settingsB.alpha=0

	learnTextFon = display.newRect(topGroup,C_X/3,870,_X/3,150)
	learnTextFon:setFillColor(0.61, 0.86, 0.32)
	learnText = display.newText(topGroup, "Learn", C_X/3, learnTextFon.y,
font, 100)

	homeTextFon = display.newRect(topGroup,C_X,870,_X/3,150)
	homeTextFon:setFillColor(0.61, 0.86, 0.32)
	homeText = display.newText(topGroup, "Home", C_X, 870, font, 100)
```

```
	testTextFon = display.newRect(topGroup,_X- C_X/3,870,_X/3,150)
	testTextFon:setFillColor(0.61, 0.86, 0.32)
	testText = display.newText(topGroup, "Tests", _X- C_X/3, 870, font,
100)

	tabLine = display.newRect(topGroup, C_X, 950, _X/3,20)

	menuBFon = display.newRoundedRect(menuGroup, 0, 170, 500,120 , 20)
	menuBFon:setFillColor(0.95,0.95,0.95)

	menuBText = display.newText(menuGroup,
"DarkSide",0,menuBFon.y,font, 75)
	menuBText:setFillColor(0,0,0)

	fillUpdateGroup()

	timer.performWithDelay( 2000, fillUpdateGroup, 0 )

	upDateIco:addEventListener("tap", checkUpdate)
	Runtime:addEventListener( "key", onKeyEvent );
	learnTextFon:addEventListener("tap",gotoLearn)
	homeTextFon:addEventListener("tap",gotoHome)
	testTextFon:addEventListener("tap",gotoTests)
	Baton:addEventListener("tap",gotoPazzle)
	Batonka:addEventListener("tap",gotoPolygon)
	settingsB:addEventListener("tap", seeMenu)
	menuBFon:addEventListener("tap", gotoDarkSide)
```

```
end

function scene:show( event )
	local sceneGroup = self.view
	local phase = event.phase
	if ( phase == "will" ) then
	elseif ( phase == "did" ) then
	end
end

function scene:hide( event )
	local sceneGroup = self.view
	local phase = event.phase
	if ( phase == "will" ) then
	elseif ( phase == "did" ) then
		learnTextFon:removeEventListener("tap",gotoLearn)
		homeTextFon:removeEventListener("tap",gotoHome)
		testTextFon:removeEventListener("tap",gotoTests)
		Baton:removeEventListener("tap",gotoPazzle)
		Batonka:removeEventListener("tap",gotoPolygon)
		Runtime:removeEventListener( "key", onKeyEvent )
		settingsB:removeEventListener("tap", seeMenu)
		composer.removeScene( currScene )
	end
end

function scene:destroy( event )
```

```
        local sceneGroup = self.view
end

scene:addEventListener( "create", scene )
scene:addEventListener( "show", scene )
scene:addEventListener( "hide", scene )
scene:addEventListener( "destroy", scene )

return scene
```