ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до магістерської роботи

на ступінь вищої освіти магістр

на тему «**РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ РОБОТИ ШВИДКОЇ МЕДИЧНОЇ ДОПОМОГИ**»

Виконав: студент 7 курсу, групи ППЗМ-71

спеціальності

6.050103 Програмна інженерія

(шифр і назва спеціальності)

Левкуша О. В.

(прізвище та ініціали)

Керівник Жебка В. В.

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Нормоконтроль Трінтіна Н.А.

(прізвище та ініціали)

Київ – 2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Магістр»
Спеціальність - 6.050103 Програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
інженерії програмного забезпечення
О. В. Негоденко
"____" _____2021 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Левкуша Олександр Віталійович
(прізвище, ім'я, по батькові)

1.Тема роботи: «Розробка інформаційної системи підвищення ефективності роботи швидкої медичної допомоги»
Керівник роботи Жебка В. В доцент кафедри кандидат технічних наук,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "12.03" 2021 року № 65.
2.Строк подання студентом роботи

3. Вихідні дані до роботи:

API для прискорення роботи працівників швидкої
Android додаток для працівників мобільних бригад

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області та постановка задачі інформаційної системи для підвищення ефективності роботи швидкої медичної допомоги
2. Теоретичні основи для розробки програмних засобів
3. Розробка програмного забезпечення. Створення API та Android додатку
4. Огляд програмної реалізації та тестування програмного забезпечення

5. Перелік графічного матеріалу
1. Мета роботи.
2. Постановка задачі.
3. Загальна структура системи.
4. Вигляд форми 109/о
5. Вигляд форми 110/о
6. Вибір засобів розробки та проектування.
7. Проектування API
8. Головна сторінка Диспетчера прийому
9. Головна сторінка Диспетчера направлення
10. Особистий кабінет лікаря мобільної бригади
11. Відкриття форми 110/о для заповнення
12. Висновки

6. Дата видачі завдання 19.04.2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської Роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	19.04.21	Виконав
2	Опис предметної області	21.04.21	Виконав
3	Постановка задачі	28.04.21	Виконав
4	Вибір методів розповсюдження інформації в соціальних мережах	02.05.21	Виконав
5	Програмна реалізація	17.05.21	Виконав
6	Тестування розробленого ПЗ	19.05.21	Виконав
7	Вступ, висновки, реферат	20.05.21	Виконав
8	Розробка обов'язкових демонстраційних матеріалів	21.05.21	Виконав
9	Попередній захист роботи	27.05.21	Виконав
10	Здача роботи в деканат	01.06.21	Виконав

Студент _____ Левкуша О. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Жебка В. В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина магістерської роботи 77 с., 35 рис., 4 табл, 23 джерел.

Об'єкт дослідження – аналіз роботи швидкої медичної допомоги.

Предмет дослідження – інформаційна система для підвищення ефективності роботи швидкої медичної допомоги та автоматизації документообігу.

Мета роботи – створення інформаційної системи для підвищення ефективності роботи швидкої медичної допомоги та автоматизації документообігу шляхом розробки прикладного програмного інтерфейсу та android додатку.

Методи дослідження - порівняльний та системний аналіз, узагальнення, моделювання, тестування, вивчення практичних розробок і документації.

В роботі проведений аналіз роботи працівників швидкої медичної допомоги, а саме диспетчерів прийому, диспетчерів направлення та працівників мобільної бригади.

На підставі аналізу інформації були сформовані вимоги до розроблення інформаційної системи та створено її загальну структуру, яка включає в себе прикладний програмний інтерфейс який має декілька ролей. Серед яких Web-інтерфейси диспетчера прийому та диспетчера направлення, а також мобільній android додаток для працівників мобільних бригад.

Було проведено концептуальне і логічне проектування прикладного програмного інтерфейсу.

Для створення інформаційної системи використовувались такі програми: WebStorm (для написання pug та js файлів які були структурою API) та PhpStorm (для написання php файлів) від компанії JetBrains, Adobe Photoshop (для створення унікального шаблону web-інтерфейсів), система управління базою даних Navicat Premium (для створення бази даних та написання автоматичних запитів до неї), інтегроване середовище розробки Android Studio (для створення Android додатку).

Актуальність роботи – дана інформаційна система орієнтована насамперед на медичних працівників швидкої та керівництва, допомагає оперативно

передавати дані та оперувати ними, а також автоматично створювати різні звіти за потрібний період.

Ключові слова: API, ANDROID ДОДАТОК, ASTERISK СЕРВЕР, JAVA, JAVASCRIPT, MYSQL, ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ РОЗРОБКИ, ДИСПЕТЧЕР НАПРАВЛЕННЯ, ДИСПЕТЧЕР ПРИЙОМУ, ПРИКЛАДНИЙ ПРОГРАМНИЙ ІНТЕРФЕЙС, ПРОГРАМНА ПЛАТФОРМА, ФОРМА 109/О, ФОРМА 110/О, ШВИДКА МЕДИЧНА ДОПОМОГА.

ЗМІСТ

ВСТУП 9
1 ЗАГАЛЬНА ЧАСТИНА 11
1.1 Постановка задачі 11
1.2 Конструктивна частина 11
1.2.1 Поняття WEB-сайт 11
1.2.2 Особливості роботи з WEB проектами 12
1.2.3 Поняття API 13
1.2.4 Архітектура WEB сайту 15
1.2.5 Клієнтська частина 22
1.2.6 Серверні технології 25
1.2.7 Програмна платформа Node.js 26
1.3 Опис мов програмування які застосовувались 27
1.4 База даних 39
2 СПЕЦІАЛЬНА ЧАСТИНА 42
2.1 Загальна структура системи 42
2.2 Загальна структура бази даних 43
2.3 Технічне завдання для реалізації диспетчера прийому 47
2.4 Технічне завдання для реалізації диспетчера направлення 52
2.5 Технічне завдання для реалізації дій працівника бригади ШМД 56
2.6 Текст програми 59
3 СТВОРЕННЯ API ТА ANDROID ДОДАТКУ 60
3.1 Аналіз вхідних даних. Стратегія створення API 60
3.2 Аналіз вхідних даних. Стратегія створення Android додатку 63
3.3 Інтеграція Asterisk з API 69
3.4 Інструменти для розробки 72
3.5 Оцінка якості 83
ВИСНОВКИ 84
ПЕРЕЛІК ПОСИЛАНЬ 86
ДОДАТОК А. Програмний код API інтерфейсу 90
ДОДАТОК Б. Програмний код Android додатку 100
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 105

ВСТУП

На сучасному етапі розвитку суспільства одними з найважливіших напрямків є інформаційні технології. З кожним роком обсяг інформації незмінно збільшується, змушуючи витрачати на свою обробку все більшу кількість часу і трудових ресурсів. У зв'язку з цим все більш необхідними стають сучасні автоматизовані інформаційні системи, здатні за малі терміни обробляти вхідну інформацію і надавати її в зручному вигляді.

До таких інформаційних систем в першу чергу варто віднести автоматизовані системи управління, головним елементом яких є бази даних, що дозволяють раціонально і доступно зберігати використовувану інформацію.

Впровадження інформаційних технологій почали використовувати також в медичних закладах. Для цього потрібно перейти на більш сучасний рівень комп’ютерних технологій. Саме комп’ютерні технології сприяють підвищенню швидкості управління персоналом та підготовкою паперової документації.

Автоматизація системи Швидкої медичної допомоги на сьогоднішній день є одним з критеріїв сучасної країни. Вона дозволяє максимально ефективно використовувати весь потенціал персоналу який відповідає на дзвінки громадян, та лікарів які виїжджають на місце пригоди, або додому до пацієнта.

З використанням автоматизованої системи Швидкої медичної допомоги ми перекладаємо на її плечі всю рутинну роботу з якою вона чудово та швидко справляється. Нам залишається лише управляти цією системою.

Об'єктом дослідження даної роботи є діяльність Швидкої медичної допомоги. Предмет дослідження - процес надання допомоги громадянину який звернувся за допомогою до Швидкої медичної допомоги.

Метою цієї магістерської роботи є розробка інформаційної системи, яка дозволить підвищити ефективність роботи Швидкої медичної допомоги.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз і зробити загальну характеристику предметної області та об'єкту автоматизації;

- розробити вимоги до інформаційної системи;
- провести аналіз існуючих інформаційних систем;
- провести обґрунтування і вибір засобів розробки і проектування
- спроектувати структуру бази даних;
- реалізувати інформаційну систему.

Методи дослідження і розробки проекту - порівняльний та системний аналіз, узагальнення, моделювання, тестування, вивчення практичних розробок і документації.

Практична значимість проекту – розроблене робоче місце дозволить впровадити електронний облік та деяку автоматизацію Швидкої медичної допомоги, підвищити тим самим, ефективність та швидкість обліку, знизити навантаження на працівників чергової частини та забезпечити ефективний інструмент контролю для керівного складу.

1 ЗАГАЛЬНА ЧАСТИНА

1.1 Постановка задачі

Початок роботи над проектом завжди передбачає перед собою розробку технічного завдання за участю інженера-програміста та замовником проекту і даний проект не став винятком з правил. Замовником програмного забезпечення стало «Міністерство охорони здоров’я», міністерство вже свої сайти та API для автоматизації роботи в державних лікарнях. Тому основні аспекти на які замовник загострив увагу були в інтеграції інформаційної системи з IP-телефонією.

1.2 Конструктивна частина

1.2.1 Поняття WEB-сайт

Інформація, доступна користувачам Internet, розташовується на комп'ютерах (веб-серверах), на яких встановлено спеціальне програмне забезпечення. Значна частина цієї інформації організована у вигляді веб-сайтів. Кожен з них має своє ім'я (адреса) в Internet.

Веб-сайт – це інформація, представлена в певному вигляді, яка розташовується на веб-сервері і має своє ім'я (адреса). Для перегляду веб-сайтів на комп'ютері користувача використовуються спеціальні програми, які називаються браузерами. Залежно від того, яке ім'я (адреса) сайту ми задаємо в рядку "Адреса", браузер завантажуватиме в своє вікно відповідну інформацію.

Веб-сайт складається із зв'язаних між собою веб-сторінок. Веб-сторінка є текстовим файлом з розширенням *.html, який містить текстову інформацію і спеціальні команди – HTML-код, що визначають в якому вигляді ця інформація відображатиметься у вікні браузера. Вся графічна, аудіо - і відеоінформація безпосередньо в Веб-сторінку не входить і є окремими файлами з розширеннями

*.gif, *.jpg (графіка), *.mid, *.mp3 (звук), *.avi (відео). У HTML-коді сторінки містяться тільки вказівки на такі файли.

Кожна сторінка веб-сайту також має свій Internet адрес, який складається з адреси сайту та імені файлу, відповідного даній сторінці. Таким чином, веб-сайт – це інформаційний ресурс, що складається із зв'язаних між собою гіпертекстових документів (веб-сторінок), розміщений на веб-сервері і такий, що має індивідуальну адресу. Подивитися веб-сайт може будь-яка людина, що має комп'ютер, підключений до Internet [10].

1.2.2 Особливості роботи з WEB проектами

Веб-додаток - додаток клієнт-серверної архітектури, в якій логіка розподілена між клієнтом (браузером) та сервером (веб-сервером). Збереження даних, як правило, переважно відбувається на сервері, і обмін даними відбувається по мережі. Завдяки клієнт-серверній архітектурі веб-додатки не залежать від платформи, на якій вони запускаються.

Основна логіка роботи програми знаходиться на сервері, який за запитом клієнта формує веб-сторінку і відправляє її для перегляду. Для програмування серверної частини веб-додатки використовуються різні мови програмування, найпопулярніші це PHP, Python, Ruby, Java, C #, Perl та інші.

Якщо веб-додаток потребує зберігання будь-яких даних, то використовують різні СУБД, такі як MySQL, MariaDB, MS SQL Server та інші, звернення до яких можливо за допомогою не процедурної мови програмування SQL. Також, крім SQL баз даних, можуть застосовуватися так звані NoSQL бази даних, такі як MongoDB, CouchDB, Couchbase, MarkLogic і інші.

Їх основні відмінності полягають в тому, що вони не використовують SQL і дані в них не структуровані. Клієнт, відправляючи запит серверу у відповідь отримує веб-сторінку, яку відображає браузер.

Сервер генерує сторінку на мові гіпертекстової розмітки HTML (XHTML), який інтерпретується браузерами і відображає її в зручному для перегляду вигляді.

З метою зміни зовнішнього вигляду сторінок, завдання стилю використовується мова каскадних таблиць стилів CSS. На стороні клієнта так само можуть зберігатися дані у вигляді файлів, необхідних для відображення веб-сторінки. Така технологія називається cookie. Цю технологію частіше використовують для аутентифікації користувачів, зберігання персональних налаштувань та ведення статистики про користувачів. Інформацію, що зберігається в cookie можуть використовувати різні скрипти, які виконуються на пристрої клієнта.

Самим популярними засобами для написання клієнтських скриптів є мова програмування JavaScript і безліч його фреймворків (jQuery, AngularJS, Backbone.js, Ember.js і інші). Скрипти необхідні для виконання певної логіки роботи веб-додатки, яка може бути виконана на стороні клієнта, що знижує навантаження на сервер. Також вони необхідні для опрацювання деяких елементів інтерфейсу [10].

1.2.3 Поняття API

Прикладний програмний інтерфейс (англ. Application Programming Interface, API) — набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено - це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

Одним з найпоширеніших призначень API є надання набору широко використовуваних функцій, наприклад для малювання вікна чи іконок на екрані. Програмісти використовують переваги API у функціональності, таким чином їм не доводиться розробляти все з нуля. API є абстрактним поняттям — програмне забезпечення, що пропонує деякий API, часто називають реалізацією (англ. implementation) даного API. У багатьох випадках API є частиною набору розробки програмного забезпечення, водночас, набір розробки може включати як API, так і

інші інструменти чи апаратне забезпечення, отже ці два терміни не є взаємозамінювані.

Високорівневі API часто програють у гнучкості. Виконання деяких функцій нижчого рівня стає набагато складнішим, або навіть неможливим.

В об'єктно-орієнтованих мовах, прикладний програмний інтерфейс зазвичай включає в себе опис набору визначень класу, з набором форм поведінки, пов'язаних з цими класами. Це абстрактне поняття пов'язане з реальними функціями, які надані або надаватимуться, класами, які реалізуються в методах класу.

Прикладний програмний інтерфейс в даному випадку можна розглядати як сукупність всіх методів, які публічно доступні в класах (зазвичай званий інтерфейс класу). Це означає, що прикладний програмний інтерфейс вказує методи, за допомогою яких взаємодіє з об'єктами, отриманими з визначень класів і обробляє їх.

У більш загальному плані можна визначити Прикладний Програмний Інтерфейс як сукупність усіх видів об'єктів, які можна вивести з визначення класу, і пов'язаних з ними можливих варіантів поведінки.

Наприклад: клас, що представляє Stack, може просто виставити публічно два методи Push() (для додавання нового елемента в стек) і Pop() (для вилучення останнього пункту, ідеально розташований на вершині стека).

У цьому випадку Прикладний Програмний Інтерфейс може бути інтерпретованим як два методи pop() і push(), або, більш широко, використовується варіант, коли можна використовувати елемент типу Stack, який реалізує поведінку стека, надаючи йому можливість для додавання / видалення елементів з вершини. Друга інтерпретація видається більш доречною в дусі об'єктно-орієнтованого підходу.

Якість документації, пов'язаної з Прикладним Програмним Інтерфейсом, є часто ключовим фактором, що визначає його успішність з точки зору простоти використання.

Бібліотеки і платформи прикладних програмних інтерфейсів як правило, пов'язані із бібліотеками програмного забезпечення: ППІ описує і вказує очікувану поведінку в той час, як бібліотека є фактичною реалізацією даного набору правил. Один ППІ може мати декілька реалізацій (або жодної, будучи абстрактним) у вигляді різних бібліотек, які мають такий же інтерфейс.

Прикладний програмний інтерфейс також може бути пов'язаним з платформами програмування: платформа може бути заснована на кількох бібліотеках реалізує декілька інтерфейсів ППІ, але на відміну від звичайного використання ППІ, доступ до поведінки вбудований в платформу опосередкований шляхом розширення його змісту новими класами і вставлений в саму платформу. Крім того, загальний потік управління програми може бути під контролем абонента [17].

1.2.4 Архітектура WEB сайту

Архітектура сайту (рис. 1.1) має на увазі наявність ряду комп'ютерів, об'єднаних між собою в мережу, один з цих комп'ютерів називається сервером (він виконує спеціальні функції), всі інші - клієнтами.

Дана архітектура розділяє додаток на клієнтське і серверне. Додаток-клієнт формує запит до сервера, де розташована база даних, на структурному мовою запитів SQL. Віддалений сервер (в нашому випадку це локальний сервер - Apache) приймає запит і адресує його SQL-серверу бази даних, саме на віддаленому сервері виконуються всі логічні скрипти, що виконуються на php. SQL-сервер - особлива програма, яка керує віддаленою базою даних.

SQL-сервер забезпечує інтерпретацію запиту, його виконання в базі даних, формування результату виконання запиту та видачу його додатку-клієнту. При цьому ресурси на клієнтському комп'ютері не беруть участь у фізичному виконанні запиту; клієнтський комп'ютер лише відсилає запит до серверної бази даних і отримує результат, після цього обробляє його необхідним чином і виводить користувачеві.

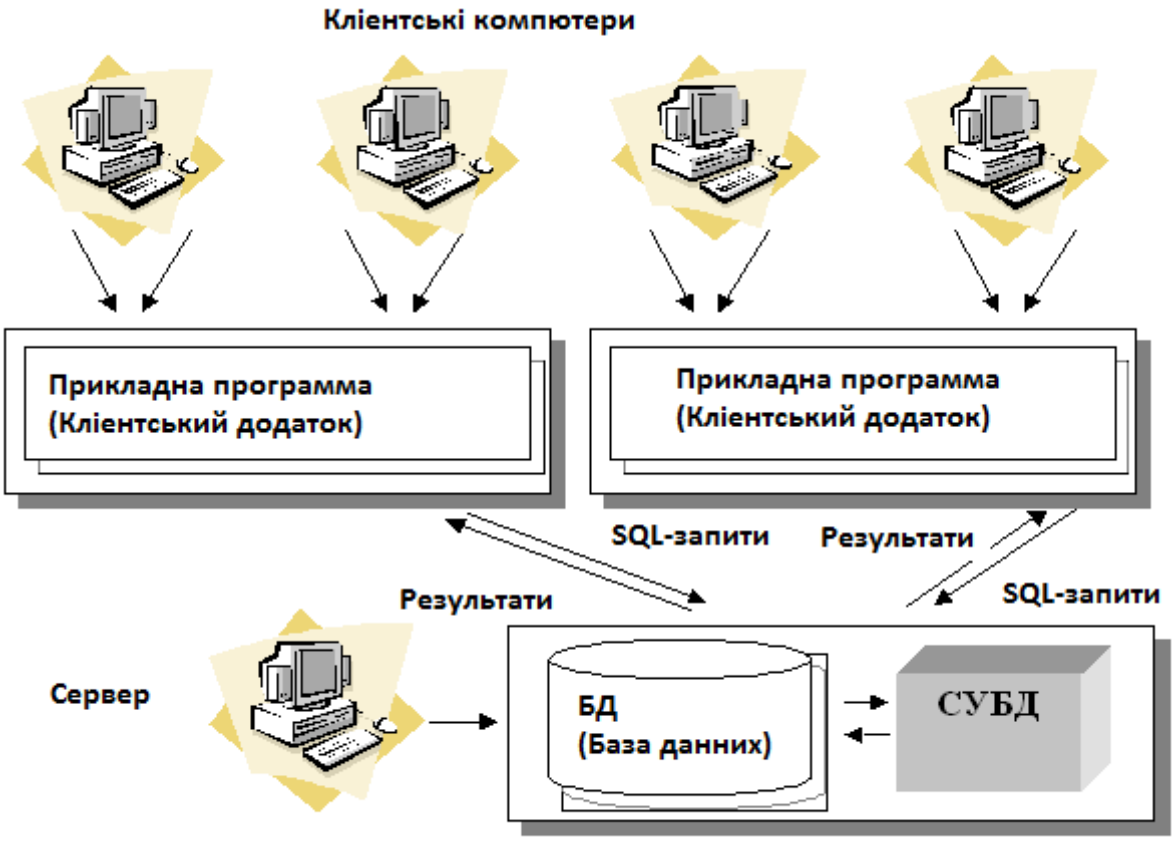


Рисунок 1.1 — Архітектура «клієнт сервер»

Так само на клієнтській стороні виконуються клієнтські скрипти, зокрема в даному проекті jQuery-бібліотека. Так як клієнтському додатку посилається результат виконання запиту, по мережі "подорожують" тільки ті дані, які необхідні клієнту. В результаті знижується навантаження на мережу. Так як виконання запиту відбувається там же, де зберігаються дані (на сервері), не потрібно пересилати великі пакети даних. Так само, SQL-сервер оптимізує отриманий запит так, щоб він був виконаний за мінімальний час.

Робота архітектури "клієнт - сервер" побудована таким чином:

• база даних знаходиться на жорсткому диску, на спеціально виділеному комп'ютері (сервера мережі);

•СУБД розташовується на сервері;

•існує локальна мережа, що складається з клієнтських комп'ютерів, де на кожному встановлено клієнтське додаток, що працює з базою даних;

- на кожному з клієнтських комп'ютерів користувачі мають можливість запустити додаток. Використовуючи наданий призначений для користувача інтерфейс, він ініціює звернення до СУБД, на вибірку / оновлення інформації. Для спілкування використовується спеціальна мова запитів SQL, тобто по мережі від клієнта до сервера передається лише текст запиту;
- СУБД містить всі відомості про фізичну структуру бази даних;
- СУБД ініціює звернення до даних, що знаходяться на сервері, в результаті яких на сервері здійснюється вся обробка даних і лише результат виконання запиту копіюється на клієнтський комп'ютер. Таким чином СУБД повертає результат в додаток;
- додаток, використовуючи призначений для користувача інтерфейс, відображає результат виконання запитів.

Розмежування функцій між сервером і клієнтом:

1. Функції програми-клієнта:
   - посилка запитів серверу;
   - інтерпретація результатів запитів, отриманих від сервера;
   - представлення результатів користувачеві в певній формі (інтерфейс користувача).
2. Функції серверної частини:
   - прийом запитів від програм-клієнтів;
   - інтерпретація запитів;
   - оптимізація і виконання запитів до бази даних;
   - відправка результатів додатку-клієнта;
   - забезпечення системи безпеки і розмежування доступу;
   - управління цілісністю бази даних.

Для опису розробки інформаційної системи потрібно побудувати модель. Найбільш зручним мовою моделювання є IDEF0, запропонований Дугласом Россом.

Метод функціонального моделювання IDEF0 (раніше називався SADT) - метод структурного аналізу і проектування.

Даний метод являє сукупність правил і процедур, призначених для побудови функціональної моделі об'єкта будь-якої предметної області. Функціональна модель SADT відображає функціональну структуру об'єкта, тобто вироблені їм дії і зв'язку між цими діями [15].

Результатом застосування моделі SADT є модель, яка складається з діаграм, фрагментів текстів. Графічна діаграма - головний компонент IDEF0-моделі, що містить блоки, стрілки, з'єднання блоків і стрілок і асоційовані з ними відносини.

На (рис. 1.2) модель IDEF0, що описує основні етапи створення інтернет-вузла.

На (рис. 1.4) представлена деталізована діаграма IDEF0.

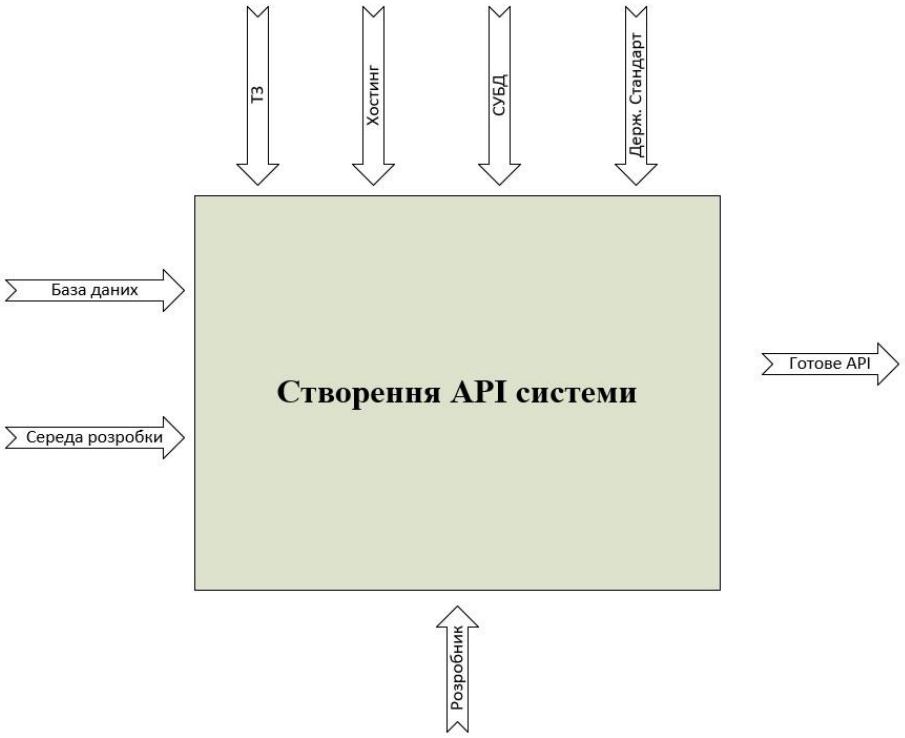


Рисунок 1.2 — IDEF0-діаграма (контекстна)

Після опису системи в цілому проводиться розбиття її на великі фрагменти, цей процес називається функціональної декомпозицією, а діаграми, які описують кожен фрагмент і взаємодія фрагментів, називаються діаграмами декомпозиції. На

(рис. 1.4) представлена деталізація, не всіх процесів розробки, а тільки частини, а саме процесу розробки дизайну та підключення системи.

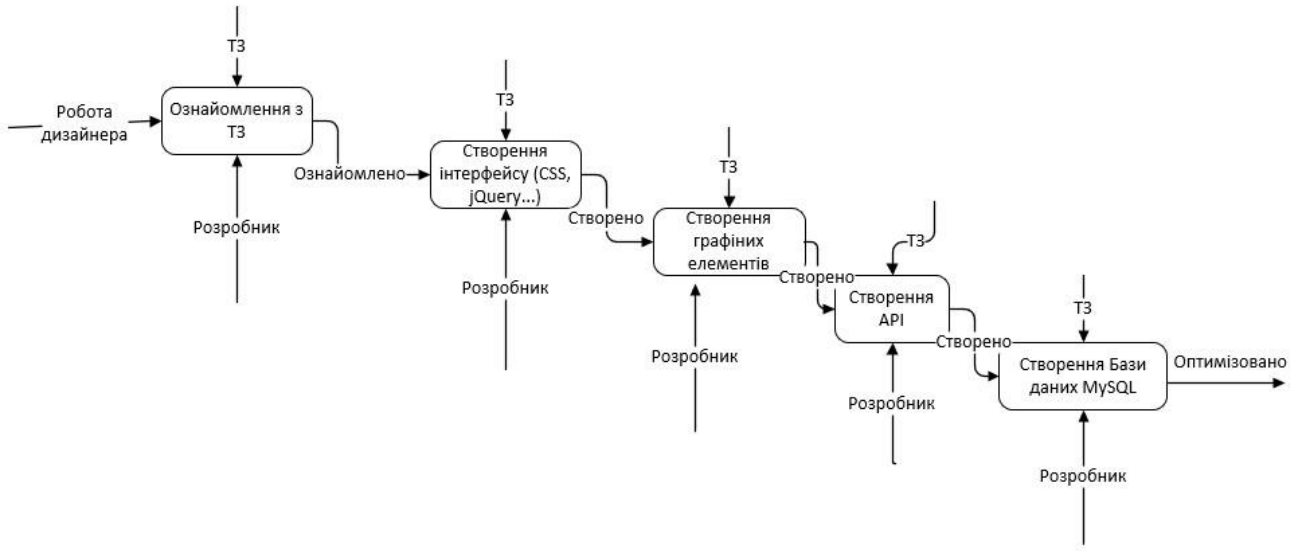


Рисунок 1.3 – IDEF0-діаграма (деталізація процесу розробки дизайну)

Модель IDEF3

Основою моделі IDEF3 служить сценарій процесу, який виділяє послідовність дій і процесів аналізованої системи. Основною одиницею моделі є діаграма. Іншою важливою компонентною є дія.

Все зв'язки в IDEF3 є односпрямованими.

На (рис. 1.5) представлена модель IDEF3.

DEF0 — Function Modeling — методологія функціонального моделювання і графічного описання процесів, призначена для формалізації і опису бізнес-процесів. Особливістю IDEF0 є її акцент на ієрархічне представлення об'єктів, що значно полегшує розуміння предметної області. В IDEF0 розглядаються логічні зв'язки між роботами, а не послідовність їх виконання в часі (WorkFlow).

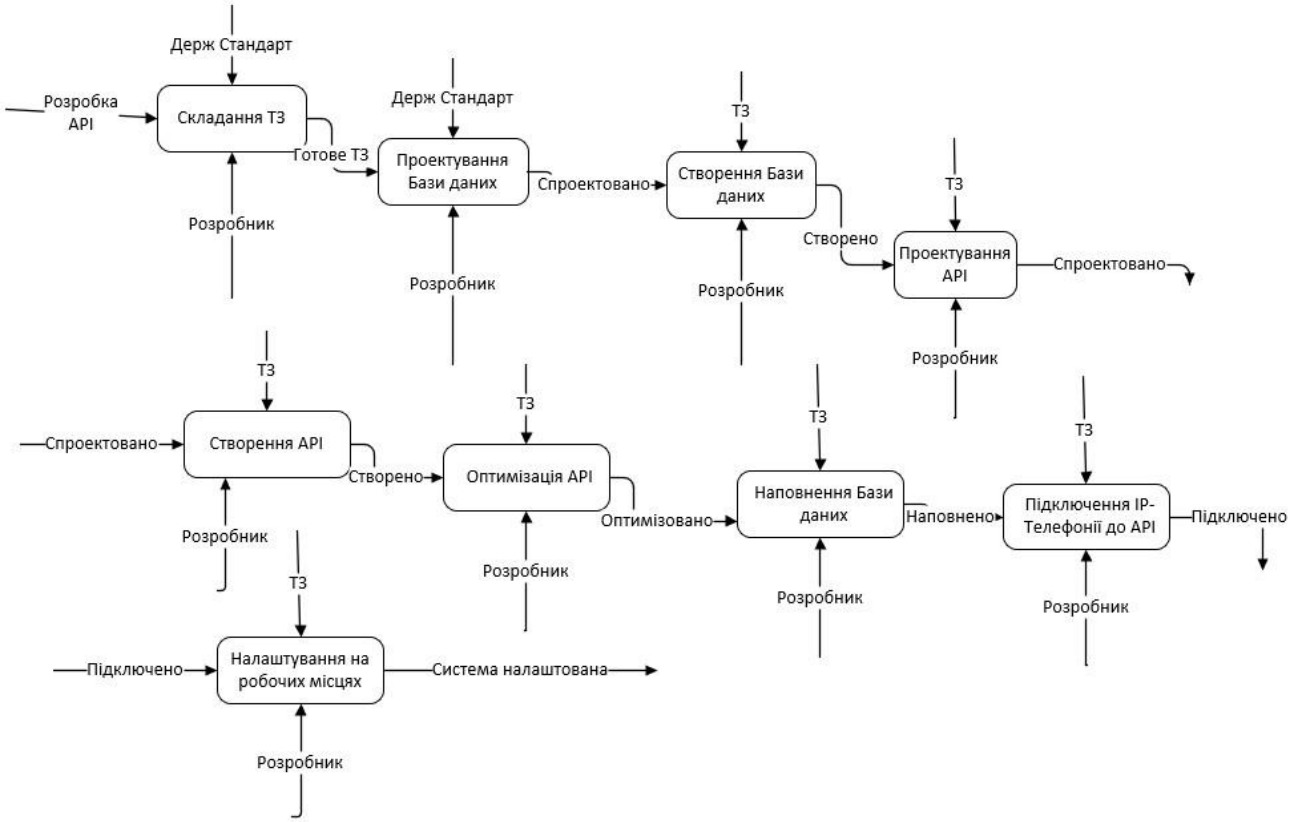


Рисунок 1.4 – IDEF0-діаграма (детальна)

Так само відображаються всі сигнали управління. Така модель є однією з найпрогресивніших моделей і використовується в організації бізнес проектів і проектів, що базуються на моделюванні всіх процесів як адміністративних, так і організаційних.

DFD-модель

На даному етапі проектування інформаційної системи будується діаграма потоків даних (DFD).

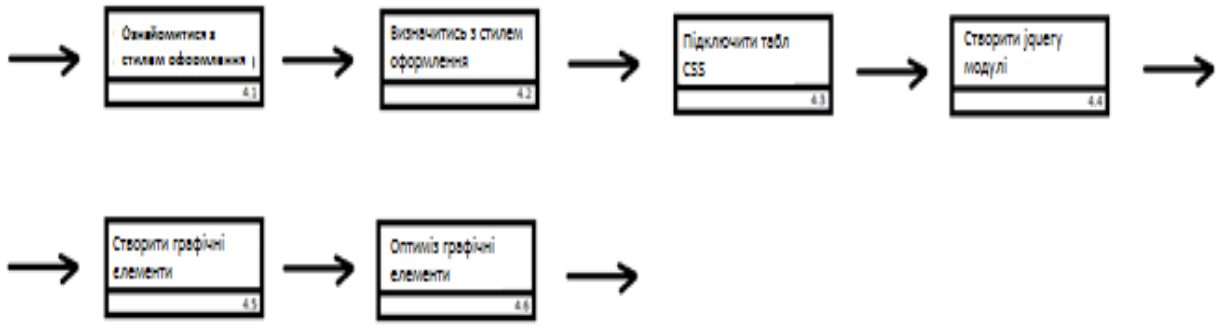


Рисунок 1.5 – Модель IDEF3 деталізуюча процес розробки дизайну сайту

DFD - діаграма (рис. 1.6) потоків даних являє собою ієрархію функціональних процесів, пов'язаних потоками даних. Мета такого уявлення - продемонструвати, як кожен процес перетворює свої вхідні дані у вихідні, а також виявити ставлення між цими процесами. Для побудови DFD-моделі традиційно використовують 2 нотації: Йордана-де Марка і Гейне Сарсона. У даній роботі використовується нотація Гейне Сарсона при побудові DFD-моделі. Відповідно до даного методу модель системи визначається як ієрархія діаграм потоків даних, що описують процес перетворення інформації від її введення в систему до видачі споживачеві.

Основними компонентами DFD-моделі є зовнішні сутності, системи та підсистеми, процеси, накопичувачі даних і потоки даних.

Зовнішня сутність є матеріальним об'єктом, джерело або приймач інформації.

Процес являє собою перетворення вхідних потоків даних у вихідні відповідно до певного алгоритму.

Накопичувач даних - абстрактне пристрій зберігання інформації, яку можна в будь-який момент перемістити в накопичувач даних і через деякий час витягнути.

Потоки даних визначають інформацію, передану через деякий з'єднання від джерела до приймача.

1.2.5 Клієнтська частина

HTML5-нова відкрита платформа, призначена для створення веб-додатків використовують аудіо, відео, графіку, анімацію та ін. HTML5 вводить цілий ряд нових елементів, які спрощують структуру сторінок. Більшість сторінок на HTML4 містять типові елементи, такі як «шапка», «підвал» і колонки. Нині, як правило, в коді документа вони позначаються за допомогою елементів , описуючи кожен атрибутом id або class.

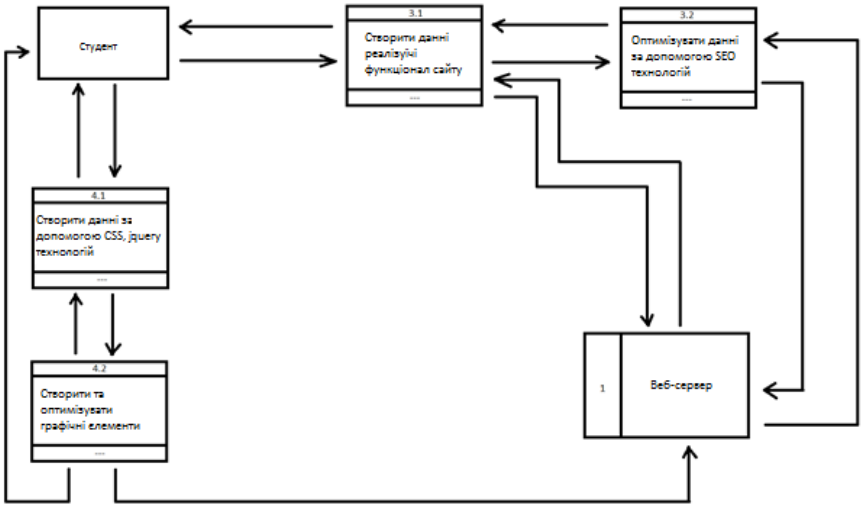


Рисунок 1.6 – DFD модель

HTML5 залишився мовою розмітки, але відрізняється від попередньої версії досить серйозно. Якщо минула четверта версія була тільки мовою розмітки, то нова версія містить мову розмітки, засоби програмування CANVAS на базі JavaScript, розширені можливості роботи з медіа, розширені елементи форм.

Дану мову взагалі називають мовою семантичної розмітки через чітку структуру представлення елементів сайту, так як в даній версії замість єдиного тега визначає блок, з'явилася можливість розбиття статті на логічні області за допомогою нових тегів <HEADER><BODY><FOOTER> і семантичного розбиття <SECTION><ARTICLE><DIV><ASIDE><NAV>.

Починаючи з 2010 року в якості стандартів була запропонована наступна зв'язка HTML5 + CSS3 + JavaScript. У сучасній розробці клієнтської частини сайту на етапі розмітки, тобто побудови макета сторінок сайту використовується HTML5 в зв'язці з таблицями стилів CSS3. Розширені таблиці стилів CSS3 - це новий стандарт оформлення HTML документів, що значно розширює можливості попереднього стандарту CSS2.

Ці стилі дозволяють створювати візуальні ефекти і навіть анімацію, причому досить високого рівня. Короткий опис нових можливостей:

- створення елементів зі згладженими кутами;
- створення лінійних і сферичних градієнтів;
- гнучке оформлення фонових картинок елементів;
- додавання до елементів і до тексту тіні;
- створення анімацій і різних ефектів переходів;
- створення різних трансформацій;
- завдання квітів декількома новими способами;
- оформлення кордонів, фонів, тексту.

Найбільш вузьким місцем при роботі з API є його постійний зв'язок з сервером. До недавнього часу було чітке розділення - оформленням займається клієнтська частина сайту, а функціоналом - серверна. Але кількість користувачів як мережі загалом, так і окремих сайтів, зокрема, стає дедалі більшою. Більш того, зростання кількості користувачів системою є однією з кінцевих цілей розробки даної системи, тому щоб зняти навантаження на сервер, частина функцій сайту, в сучасній веб розробці, передається на сторону клієнта. Для цих цілей розроблено декілька технологій, найбільш розвинута з яких - AJAX (асинхронний JavaScript і XML).

Мета даної технології реалізація деяких функцій сайту без звернення до сервера. Для створення динамічного контенту і реалізації клієнтської частини функціоналу сайту використовуються клієнтські мови програмування [10].

Як стандарти мови клієнтського веб програмування визначено JavaScript. Однією з кардинальних особливостей клієнтських мов програмування полягає в тому, що вони виконуються браузером, і вони не мають прямого доступу до файлової системи, а значить і до баз даних (є винятки для XML файлів). Це стосується і JavaScript.

Дана мова є об'єктним, об'єктно-орієнтованим і, як об'єктна мова підтримує DOM (об'єктну технологію подання документа або подання веб документа у вигляді групи об'єктів на деякій підкладці, яка носить назву window. Вікно може бути тільки одне і воно має низку властивостей. В вікні розташовується документ (який так само має ряд властивостей), а вже на документі розташовуються елементи в певному порядку. Порядок розташування елементів може бути порушений при використанні CSS стилів, які, в свою чергу, можуть управлятися командами JavaScript.

Найбільш поширеним сьогодні є фреймворк jQuery с фреймворком всередині себе jQueryUI. jQuery дозволяє виконувати всі функції клієнтської веб розробки і значно розширює і спрощує код завдяки розробленим в ньому функцій. jQuery UI побудований на базі фреймворка і спрямований на оформлення основних елементів інтерфейсної частини сайту (меню, текстові блоки, галереї і т.д.). Використання фреймворків в розробці є характерною ознакою сучасного сайту і, в даний час, є необхідним елементом. На базі jQuery розроблений новий фреймворк керуючий динамічним оформленням клієнтської частини сайту Bootstrap - популярний фреймворк, який дозволяє швидко і якісно створювати статичні веб-сайти і веб-додатки . По суті, це безкоштовний набір інструментів, що дозволяє використовувати Html, CSS і JavaScript «великими мазками» [5].

Вимоги до клієнтського програмного забезпечення

Сайт повинен бути доступний для повнофункціонального перегляду за допомогою наступних браузерів:

- Mozilla Firefox 47.0 і вище;
- Google Chrome 50.0 і вище;

- Safari 5.1.7 і вище;
- MS IE 9.0 і вище;
- Opera 50.0 і вище.

1.2.6 Серверні технології

Як уже згадувалося вище, на серверну частину сайту покладається весь функціонал сайту, тому кількість мов програмування серверної частини сайту значно більше, ніж клієнтської і, крім того, немає однієї технології, яка була б визначена в якості стандарту розробки. Серед серверних мов можна виділити Ruby, ASP, JSP, Python, Perl, PHP, але найбільш поширеним (більше 5 млн. Серверів) є JavaScript на пладформі Node.js [5].

Особливістю даної платформи, саме в версії 14.8.0, є повна підтримка об'єктно-орієнтованої технології програмування, що спрощує роботу з повторного використання коду, покращує читаність коду і дозволяє працювати над створенням сайту команді розробників. На мові JavaScript написана досить велика кількість фреймворків (каркасів розробки структурних і функціональних елементів сайту).

Найбільш поширеними системами управління базами даних є MySQL, PostgreSQL, ORACLE. Ці бази даних є полегшеними, тобто мають урізаним функціоналом, достатнім для задач веб розробки.

Найбільш поширеною є MySQL. Дана база досить проста у використанні і має високу швидкість обробки запитів, що важливо при великій кількості звернень від користувачів до сервера, на якому розташовується сайт .

У back-end розробці найбільш поширеною є зв'язка Node.js+MySQL, JavaScript має достатньо коштів для організації повномасштабної роботи з базою даних. Розписувати саму мову програмування не має сенсу, більш докладно фрагменти роботи з програмним кодом будуть позначені у другій частині дипломної роботи.

При створенні даного програмного продукту була використана програмна платформа Node.js, мова програмування JavaScript. Для розробки бази даних – Navicat та система керування реляційними базами даних MySQL.

Navicat – це серія програм керування базами даних та розробки програмного забезпечення.

1.2.7 Програмна платформа Node.js

Node.js це - програмна платформа для JavaScript. Оточення Node.js включає все, що потрібно для виконання програми, написаної на JavaScript.

Раніше могли запустити JavaScript тільки в браузері, але одного разу розробники розширили його, і тепер можна запускати JS на своєму комп'ютері або сервері в якості окремого додатка. Так з'явився Node.js.

В наші дні платформа Node.js є однією з найпопулярніших платформ для побудови ефективних і масштабованих REST API's. Вона так само підходить для побудови гібридних мобільних додатків, десктопних програм і навіть для IoT.
Веб-додатки, написані слідуючи клієнт/серверній архітектурі, працюють за такою схемою - клієнт запитує потрібний ресурс у сервера і сервер відправляє ресурс у відповідь. У цій схемі сервер, відповівши на запит, перериває з'єднання. Коли сервер отримує новий запит він створює окремий потік для його обробки.

Потік, якщо простими словами, це час і ресурси, що CPU виділяє на виконання невеликого блоку інструкцій. З урахуванням сказаного, сервер може обробляти кілька запитів одночасно, але тільки по одному на потік. Така модель так само називатися thread-per-request model. Для обробки N запитів серверу потрібно N потоків. Якщо сервер отримує N+1 запитів, тоді він повинен чекати поки один з потоків не стане доступним. Серед аналогів розробки Front-end частини сайту чи API є CMS системи [18].

1.3 Опис мов програмування які застосовувались

JavaScript

JavaScript — динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки.

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

Мова JavaScript використовується для:

- написання сценаріїв веб-сторінок для надання їм інтерактивності;
- створення односторінкових веб-застосунків (React, AngularJS, Vue.js);
- програмування на стороні сервера (Node.js);
- стаціонарних застосунків (Electron, NW.js);
- мобільних застосунків (React Native, Cordova);
- сценаріїв в прикладному ПЗ (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter);
- всередині PDF-документів тощо.

Незважаючи на схожість назв, мови Java та JavaScript є двома різними мовами, що мають відмінну семантику, хоча й мають схожі риси в стандартних бібліотеках та правилах іменування. Синтаксис обох мов отриманий «у спадок» від мови C, але семантика та дизайн JavaScript є результатом впливу мов Self та Scheme.

JavaScript має низку властивостей об'єктно-орієнтованої мови, але завдяки концепції прототипів підтримка об'єктів в ній відрізняється від традиційних мов ООП. Крім того, JavaScript має ряд властивостей, притаманних функціональним мовам, — функції як об'єкти першого класу, об'єкти як списки, каррінг, анонімні функції, замикання (closures) — що додає мові додаткову гнучкість.

JavaScript має С-подібний синтаксис, але в порівнянні з мовою С має такі корінні відмінності:

- об'єкти, з можливістю інтроспекції і динамічної зміни типу через механізм прототипів;
- функції як об'єкти першого класу;
- обробка винятків;
- автоматичне приведення типів;
- автоматичне збирання сміття;
- анонімні функції.

JavaScript містить декілька вбудованих об'єктів: Global, Object, Error, Function, Array, String, Boolean, Number, Math, Date, RegExp. Крім того, JavaScript містить набір вбудованих операцій, які, грубо кажучи, не обов'язково є функціями або методами, а також набір вбудованих операторів, що управляють логікою виконання програм. Синтаксис JavaScript в основному відповідає синтаксису мови Java (тобто, зрештою, успадкований від С), але спрощений порівняно з ним, щоб зробити мову сценаріїв легкою для вивчення. Так, приміром, декларація змінної не містить її типу, властивості також не мають типів, а декларація функції може стояти в тексті програми після неї.

Історія

1995 року компанія Netscape поставила завдання вбудувати мову програмування Scheme чи «якусь схожу» в браузер Netscape. Для цього був запрошений Брендан Айк, американський розробник, що спеціалізувався на системному програмуванні. Також, для прискорення розробки, Netscape почали співробітництво з компанією Sun Microsystems.

З часом, концепція розроблюваної мови програмування була розширена до можливості використання безпосередньо в HTML-коді сторінки. Компанії мали на меті створити мову, що могла зв'язати різні частини веб-сайтів: зображень, Java-аплетів, об'єктної моделі документа.

Ця мова повинна була стати зручною для веб-дизайнерів та некваліфікованих програмістів. Робочою назвою нової мови була Mocha, яка була змінена на LiveScript в перших двох бета-версіях браузера Netscape 2.0. А дещо пізніше, користуючись популярністю бренду Java, LiveScript був перейменований на JavaScript і третя бета-версія (2.0B3) Netscape 2.0 вже вийшла з сучасною назвою. Для цього була придбана відповідна ліцензія у компанії Sun Microsystems, що володіла брендом Java.

1992 року компанією Nombas була розроблена скриптова мова програмування Cmm (англ. C-minus-minus, гра слів навколо мови C++), яка пізніше була перейменована на ScriptEase та могла вбудовуватися в веб-сторінки. Існує хибна думка, що JavaScript був створений під впливом Cmm. Насправді, Брендан Айк ніколи не чув про Cmm до того, як він створив LiveScript. Пізніше, Nombas зупинили розробку Cmm та почали використовувати JavaScript, а згодом брали участь у групі зі стандартизації JavaScript.

Стандартизація

У листопаді 1996 року Netscape заявила, що відправила JavaScript в організацію Ecma International для розгляду мови як промислового стандарту. В результаті подальшої роботи з'явилась стандартизована мова з назвою ECMAScript. У червні 1997 року, Ecma International опублікувала першу редакцію специфікації ECMA-262. Рік по тому, у червні 1998 року, щоб адаптувати специфікацію до стандарту ISO/IEC-16262, були внесені деякі зміни і випущена друга редакція. Третя редакція побачила світ в грудні 1999 року.

Четверта версія стандарту ECMAScript так і не була закінчена і четверта редакція не вийшла. Тим не менш, п'ята редакція з'явилася в грудні 2009 року.

У червні 2015 року вийшла шоста версія, починаючи з якої комітет ECMAScript прийняв рішення перейти на щорічні оновлення і нова версія

отримала назву ES2015. Вона отримала цілу низку нововведень, серед яких: об'єкт Promise для зручного асихронного виконування коду, деструктуруюче присвоювання, стрілочні функції, функції-генератори, шаблонні рядки, оператори оголошення змінних let та const тощо.

Версія ES2016 вийшла у червні 2016 року, серед нововведень оператор піднесення до степеня ** та метод Array.prototype.includes, який перевіряє, чи міститься переданий аргумент в масиві.

Версія ES2017, що вийшла в червні 2017 року і на сьогодні є актуальною версією стандарту додала можливість використання асихронних функцій, «висячих» ком в параметрах функцій, об'єкт Atomics, декількох нових методів для роботи з рядками [12].

Java

Java — об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. «Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

Java вплинула на розвиток J++, що розроблялась компанією «Microsoft». Роботу над J++ було зупинено через судовий позов «Sun Microsystems», оскільки ця мова програмування була модифікацією Java. Пізніше в новій платформі «Microsoft» .NET випустили J#, щоб полегшити міграцію програмістів J++ або Java на нову платформу. З часом нова мова програмування C# стала основною мовою платформи, перейнявши багато чого з Java. J# востаннє включався в версію Microsoft Visual Studio 2005. Мова сценаріїв JavaScript має схожу із Java назву і синтаксис, але не пов'язана із Java.

Історія

Мова програмування Java зародилася в 1991 р. в лабораторіях компанії Sun Microsystems. Розробку проєкту започаткував Джеймс Ґослінґ, сам проєкт мав назву «Green» (Зелений). Створення першої робочої версії, яка мала назву «Oak», зайняло 18 місяців. Оскільки виявилось, що ім'я Oak уже використовувалось іншою фірмою, то в результаті тривалих суперечок навколо назви нової мови з-поміж ряду запропонованих було вибрано назву Java, у 1995 р. мову було офіційно перейменовано.

Головним мотивом створення Java була потреба в мові програмування, яка б не залежала від платформи (тобто від архітектури) і яку можна було б використовувати для створення програмного забезпечення, що вбудовується в різноманітні побутові електронні прилади, такі як мобільні засоби зв'язку, пристрої дистанційного керування тощо.

Досить скоро майже всі найпопулярніші тогочасні веб-оглядачі отримали можливість запускати «безпечні» для системи Java-аплети всередині веб-сторінок. У грудні 1998 р. Sun Microsystems випустила Java 2 (спершу під назвою J2SE 1.2), де було реалізовано декілька конфігурацій для різних типів платформ. Наприклад, J2EE призначалася для створення корпоративних застосунків, а значно урізана J2ME для приладів з обмеженими ресурсами, таких як мобільні телефони. У 2006 році в маркетингових цілях версії J2 було перейменовано у Java EE, Java ME та Java SE відповідно.

13 листопада 2006 року Sun випустили більшу частину Java як вільне та відкрите програмне забезпечення згідно з умовами GNU General Public License (GPL). 8 травня 2007 корпорація закінчила процес, в результаті якого всі початкові коди Java були випущенні під GPL, за винятком невеликої частини коду, на який Sun не мала авторського права [19].

PHP

PHP — скриптовамова програмування, була створена для генерації HTML-сторінок на стороні веб-сервера. PHP є однією з найпоширеніших мов, що використовуються у сфері веб-розробок (разом із Java, .NET, Perl, Python, Ruby).

PHP підтримується переважною більшістю хостинг-провайдерів. PHP — проект відкритого програмного забезпечення.

PHP інтерпретується веб-сервером у HTML-код, який передається на сторону клієнта. На відміну від скриптової мови JavaScript, користувач не бачить PHP-коду, бо браузер отримує готовий html-код. Це є перевагою з точки зору безпеки, але погіршує інтерактивність сторінок. Але ніхто не забороняє використовувати PHP для генерування JavaScript-кодів, які виконуються вже на стороні клієнта.

PHP — мова, у код якої можна вбудовувати безпосередньо html-код сторінок, які, у свою чергу, коректно оброблюватимуться PHP-інтерпретатором. Обробник PHP просто починає виконувати код після відкриваючого тегу (<?php) і продовжує виконання до того моменту, поки не зустріне закриваючий тег (?>).

Велика різноманітність функцій PHP дає можливість уникати написання багаторядкових функцій, призначених для користувача, як це відбувається в C або Pascal.

Наявність інтерфейсів до багатьох баз даних:

- У PHP вбудовані бібліотеки для роботи з MySQL, PostgreSQL, mSQL, Oracle, dbm, Hyperware, Informix, InterBase, Sybase;
- Завдяки *стандарту відкритого інтерфейсу зв'язку з базами даних* (англ. *Open Database Connectivity Standard*, ODBC) можна підключатися до всіх баз даних, до яких існує драйвер.

Традиційність

Мова PHP здаватиметься знайомою програмістам, що працюють в різних областях. Багато конструкцій мови запозичені з C, Perl. Код PHP дуже схожий на той, який зустрічається в типових програмах мовами C або Pascal. Це помітно знижує початкові зусилля при вивченні PHP.

PHP — мова, що поєднує переваги Perl та C і спеціально спрямована на роботу в Інтернеті, мова з універсальним і зрозумілим синтаксисом. І хоча PHP є досить молодою мовою, вона здобула таку популярність серед web-програмістів, що в наш час є найпопулярнішою мовою для створення веб-застосунків (скриптів).

Наявність сирцевого коду та безкоштовність

Стратегія Open Source, і розповсюдження початкових текстів програм в масах, безсумнівно справили благотворний вплив на багато проектів, в першу чергу — Linux хоч і успіх проекту Apache сильно підкріпив позиції прихильників Open Source.

Сказане відноситься і до історії створення PHP, оскільки підтримка користувачів зі всього світу виявилася дуже важливим чинником в розвитку проекту PHP. Ухвалення стратегії Open Source і безкоштовне розповсюдження початкових текстів PHP надало неоціненну послугу користувачам. Окрім цього, користувачі PHP в усьому світі є свого роду колективною службою підтримки, і в популярних електронних конференціях можна знайти відповіді, навіть на найскладніші питання.

Ефективність

Ефективність є дуже важливим чинником у програмуванні для середовищ розрахованих на багато користувачів, до яких належить і web. Важливою перевагою PHP є те, що ця мова належить до інтерпретованих. Це дозволяє обробляти сценарії з достатньо високою швидкістю.

За деякими оцінками, більшість PHP-сценаріїв (особливо не дуже великих розмірів) обробляються швидше за аналогічні їм програми, написані на Perl. Проте хоч би що робили розробники PHP, виконавчі файли, отримані за

допомогою компіляції, працюватимуть значно швидше — в десятки, а іноді і в сотні разів. Але продуктивність PHP достатня для створення цілком серйозних веб-застосунків.

Історія

Історія PHP починається з 1995 року, коли Расмус Лердорф (англ. Rasmus Lerdorf) створив простий застосунок мовою Perl, що аналізував відвідування користувачами його резюме на веб-сайті. Потім, коли цим застосунком вже користувалися кілька чоловік, а число охочих одержати його постійно збільшувалося, Лердорф назвав своє творіння Інструменти для особистої домашньої сторінки англ. Personal Home Page Tools версія 1 і виставив для вільного завантаження. З цієї миті почався небувалий зліт популярності PHP.

Як це завжди буває, терміново було потрібне доопрацювання і нові доповнення. Для їхньої реалізації Расмус створює нову версію пакету, тепер уже написану на C. Отриманий таким чином інструмент набуває робочої назви PHP/FI (укр. Персональна Домашня сторінка / Інтерпретатор Форм, англ. Personal Home Page / Forms Interpreter), надалі він також буде відомий під назвою PHP 2. Ця версія вже більшою мірою схожа на сьогоднішній PHP. Вона мала синтаксис і спосіб іменування змінних в стилі мови Perl, можливість вбудовування PHP операторів в html-код сторінки, автоматичну інтерпретацію форм, інтеграцію з базами даних. При цьому все працювало досить швидко, оскільки PHP прикомпилювалася до веб-серверу Apache. До 1997 року PHP використовувався вже на 50,000 доменах (не більше 1 % всіх веб-серверів).

Того ж 1997 року до проекту PHP підключилися Зев Сураскі (англ. Zeev Suraski) і Енді Ґутманс (англ. Andi Gutmans). Ці студенти Техніону, одного з найкращих ізраїльських університетів, намагалися використовувати PHP/FI для одного з комерційних університетських проектів. При цьому їм довелося зіткнутися з багатьма труднощами і обмеженнями цієї технології. Вивчаючи початковий код PHP 2, Зев і Енді дійшли висновку про необхідність доопрацювання, а точніше істотної переробки PHP, особливо в плані синтаксису мови. Протягом декількох місяців вони блискуче впоралися з цим завданням.

Закінчивши роботу, Зев і Енді домовились з Расмусом про співпрацю в галузі розвитку та вдосконалення мови. З цієї миті з'являється PHP Group — група однодумців, що працюють над розвитком технології PHP. Одержаний продукт з'явився на світ 1998 року під назвою PHP 3.

При цьому головною особливість PHP 3 була можливість розширення ядра, що привернуло до роботи над PHP безліч сторонніх розробників, що створюють спеціалізовані модулі. Їх наявність дала PHP можливість працювати з величезною кількістю баз даних, протоколів, підтримувати велике число API. До кінця 1998 року кількість користувачів PHP перевищила 100 тисяч, а PHP був уже встановлений на понад 10% серверах Інтернету. У той же час значному поширенню даної мови сприяли публікації в електронній пресі та видання посібників із PHP.

Відразу ж після виходу PHP 3, Енді Гутманс і Зев Сураскі почали переробку ядра PHP. В першу чергу належало вирішити проблему підвищення продуктивності. Новий продукт, названий Zend Engine (від імен творців: Zeev і Andi), успішно справлявся з поставленим завданням і був реалізований 1999 року. Основними реалізованими ідеями є можливість компіляції сценарію у виконуваний модуль, за рахунок чого продуктивність можна було підняти на порядок.

PHP 3 та PHP 4

Зеєв Сураскі та Ані Гутманс переробили аналізатор в 1997 році і сформували базу PHP 3, змінивши назву мови на рекурсивний акронім PHP: Hypertext Preprocessor. Після цього почалося публічне тестування PHP 3, а офіційний запуск відбувся в червні 1998 року. Після цього Сураскі та Гутманс розпочали нове перекодування ядра PHP, видавши Zend Engine у 1999 році. Вони також заснували Zend Technologies в Рамат-Гані, Ізраїль.

22 травня 2000 р. Був випущений PHP 4 з підтримкою Zend Engine 1.0. Станом на серпень 2008 року ця філія досягла версії 4.4.9. PHP 4 більше не розробляється, оновлення безпеки також більше не були випущені.

PHP 5

14 липня 2004 р. Був випущений PHP 5 з новим двигуном Zend Engine II. PHP 5 включав нові функції, такі як покращена підтримка об'єктно-орієнтованого програмування, розширення PHP Data Objects (PDO) (який містить легкий і послідовний інтерфейс для доступу до баз даних) та численні поліпшення продуктивності. У 2008 році PHP 5 став єдиною стабільною версією, що розроблялася. Пізній статичний зв'язок відсутній у PHP і був доданий у версії 5.3.

Багато високопрофесійних проектів з відкритим кодом з 5 лютого 2008 року перестали підтримувати PHP 4 в новому коді, оскільки це ініціатива GoPHP5, що надається консорціумом розробників PHP, що сприяє переходу від PHP 4 до PHP5.

З часом перекладачі PHP стали доступними для більшості існуючих 32-розрядних та 64-розрядних операційних систем, або будували їх з вихідного коду PHP, або використовували попередньо побудовані двонарні файли. Для версій PHP версії 5.3 та 5.4 єдиними доступними двосторонніми дистрибутивами Microsoft Windows були 32-розрядні версії x86, що вимагають 32-розрядний режим сумісності Windows при використанні інформаційних служб Інтернету (IIS) на 64-розрядній платформі Windows . PHP версії 5.5 зробив збірки 64-розрядних x86-64 доступними для Microsoft Windows.

PHP 6 та Unicode

PHP отримав змішані відгуки через відсутність власної підтримки Unicode на рівні основної мови. У 2005 році був започаткований проект, очолюваний Андрієм Змієвським, для залучення рідної підтримки Unicode на PHP, шляхом вбудовування бібліотеки "Міжнародні компоненти для Unicode" (ICU) та вбудованих текстових рядків як UTF-16. Оскільки це призведе до серйозних змін як до внутрішньої частини мови, так і до коду користувача, планувалося випустити його як версію 6.0 мови разом з іншими основними функціями, які розвиваються.

Проте дефіцит розробників, які зрозуміли необхідні зміни та проблеми продуктивності, що виникають внаслідок перетворення на UTF-16 та з нього, що

рідко використовується в веб-контексті, призвело до затримок проекту. Як результат, випуск PHP 5.3 був створений у 2009 році, при цьому багато не-Unicode-функцій було відновлено з PHP 6, зокрема простору імен. У березні 2010 року проект у своїй нинішній формі був офіційно відкинутий, і був підготовлений випуск PHP 5.4, що містить більшість опублікованих функцій, що не входять до Unicode, з PHP 6, такі як риси та переприв'язка до закриття. Початкові сподівання полягали в тому, що для інтеграції з Unicode був би сформований новий план, але з 2014 року ніхто не був прийнятий.

PHP 7

Протягом 2014 та 2015 років було розроблено нову основну версію PHP, яку було пронумеровано PHP 7. Нумерація цієї версії викликала дебати. Хоча реліз PHP 6 з Unicode ніколи не був випущений, декілька назв статей і книг посилалися на назву PHP 6, що могло викликати плутанину, якби новий реліз повторно використовував цю назву. Після голосування було обрано назву PHP 7.

Визнання та поширення

PHP 4, що працює на цьому ядрі, вийшов 2000 року. На додаток до збільшення продуктивності, PHP 4 мав нові можливості щодо підтримки сесій, буферизацію виводу, безпечні способи обробки інформації, що вводиться користувачем, і нові мовні конструкції. З виходом 4 версії PHP став використовуватися вже на більш ніж 20 % доменів Інтернету.

Протягом 2000—2004 років продовжувалися активні роботи з покращення четвертої версії, але майже відразу PHP Group приступила до продумування можливостей нової версії. В першу чергу було вирішено підсилити об'єктні можливості мови, що дозволяло використовувати його для реалізації масштабних проектів. Роботи із створення п'ятої версії велися тривалий час, в них брало участь рекордна кількість фахівців, зокрема Стерлінг Хьюз (англ. Sterling Hughes) і Маркус Бергера (англ. Marcus Boerger).

У липні 2004 року виходить офіційний реліз PHP 5. В першу чергу, як і планувалося, було перероблено весь механізм роботи з об'єктами. І якщо в попередніх версіях об'єктно-орієнтоване програмування на PHP було можливе в

мінімальному ступені, а тому і використовувалося на практиці не часто, то PHP 5 володіє прекраснимпотенціалом реалізації об'єктного програмування. Окрім цього, PHP збагатився рядом цінних розширень для роботи з XML, різними джерелами даних, генерації графіки і інше.

Серед інших украй корисних доповнень в PHP 5 слід зазначити нову схему обробки винятків. Конструкція try/catch/throw дозволяє весь код обробки помилок локалізувати в одному місці сценарію.

Всі основні бібліотеки для роботи з XML, запозичені в PHP 4, були піддані серйозній переробці. Такі популярні розширення, як SAX, DOM і XSLT, тепер використовують інструмент libxml2, що робить їх ще ефективнішими.

У PHP 5 також включені два нові модулі для роботи з протоколами — SimpleXML і SOAP. SimpleXML дозволяє значно спростити роботу з XML-даними, представляючи вміст XML-документа у вигляді PHP-об'єкта. Розширення SOAP дозволяє будувати на PHP сценарії, що обмінюються інформацією з іншими застосунками за допомогою XML-повідомлень поверх існуючих веб-протоколів, наприклад HTTP. Модуль для роботи з SOAP для PHP 5 надає розробникам засіб для достатньо швидкого створення ефективних SOAP-клієнтів і SOAP-серверів.

Новий модуль PHP 5 MySQLi (MySQL Improved) призначений для роботи з MySQL-сервером версій 4.1.2 і вище, реалізовуючи не тільки процедурний, але і об'єктно-орієнтований інтерфейс до MySQL. Додаткові можливості цього модуля включають — SSL, контроль транзакцій, підтримка реплікації та інші.

Шоста версія PHP розроблялася з жовтня 2006 року. Було зроблено безліч нововведень, як, наприклад, виключення з ядра регулярних виразів POSIX і «довгих» суперглобальних масивів, видалення директив safe_mode, magic_quotes_gpc і register_globals з конфігураційного файлу php.ini. Одним з основних нововведень повинна була стати підтримка Юнікоду. Однак у березні 2010 року розробка PHP6 була визнана безперспективною через складнощі з підтримкою Юнікоду. Вихідний код PHP6 переміщений на гілку, а основною лінією розробки стала версія 5.4.

У 2014 році було проведено голосування, за результатами якого наступна версія отримала назву PHP 7. Вихід нової версії планувався в середині жовтня 2015 року. У березні 2015 року Zend представили інфографіку в якій описані основні нововведення PHP 7 [16].

3 грудня 2015 року було оголошено про вихід PHP версії 7.0.0.

Нова версія грунтується на експериментальній гілці PHP, яка спочатку називалася phpng (PHP Next Generation - наступне покоління), і розроблялася з упором на збільшення продуктивності і зменшення споживання пам'яті. У новій версії додана можливість вказувати тип повертаються з функції даних, доданий контроль переданих типів для скалярних даних, а також нові оператори.

1.4 База даних

MySQL

MySQL — вільна система керування реляційними базами даних.

MySQL був розроблений компанією «TcX» для підвищення швидкодії обробки великих баз даних. Ця система керування базами даних (СКБД) з відкритим кодом була створена як альтернатива комерційним системам. MySQL з самого початку була дуже схожою на mSQL, проте з часом вона все розширювалася і зараз MySQL — одна з найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування.

Ліцензування

MySQL має подвійне ліцензування. MySQL може розповсюджуватися відповідно до умов ліцензії GPL. Але за умовами GPL, якщо якась програма використовує бібліотеки MySQL, то вона теж повинна розповсюджуватися за ліцензією GPL. Проте це може розходитися з планами розробників, які не бажають відкривати сирцеві тексти своїх програм.

Для таких випадків передбачена комерційна ліцензія компанії Oracle, яка також забезпечує якісну сервісну підтримку. В разі використання та розповсюдження програмного забезпечення з іншими вільними ліцензіями, такими як BSD, Apache, MIT та інші, MySQL дозволяє використання бібліотек MySQL за ліцензією GPL.

Історія

MySQL виникла як спроба застосувати mSQL до власних розробок компанії: таблиць, для яких використовувалися ISAM — підпрограми низького рівня для індексного доступу до даних. У результаті був вироблений новий SQL-інтерфейс, але API-інтерфейс залишився в спадок від mSQL. Звідки походить назва «MySQL» — достеменно не відомо. Розробники дають два варіанти: або тому, що практично всі напрацювання компанії починалися з префікса My, або на честь дівчинки на ім'я My, дочки Майкла Монті Віденіуса, одного з розробників системи.

Логотип MySQL у вигляді дельфіна носить ім'я «Sakila». Він був обраний з великого списку запропонованих користувачами «імен дельфіна». Ім'я «Sakila» було відправлено Open Source-розробником Ambrose Twebaze.

В січні-лютому 2008 Sun Microsystems придбала розробника системи керування базами данних MySQL за $1 млрд. Після поглинання у 2009 році Sun Microsystems компанією Oracle Corporation MySQL стала власністю Oracle.

За час розвитку під орудою Oracle дедалі більше відокремлює MySQL від спільноти і робить процес розробки все менш прозорим. Наприклад, повернута практика поставки власницьких розширених функцій в Enterprise-версії MySQL, спостерігається приховування інформації про вразливості, зі складу виключений тестовий набір, закритий доступ до більшої частини системи відстеження помилок та припинено публікація згрупованого логу змін, що дозволяє судити про прив'язку патчів до конкретних змін.

MySQL 5.1

Версія MySQL 5.1 продовжує шлях до стандарту SQL:2003. MySQL 5.1 містить такі нововведення:

Сегментування — можливість розбити одну велику таблицю на декілька частин, розміщених в різних файлових системах, базуючись на визначеній користувачем функції. При деяких умовах це може дати серйозне збільшення продуктивності та, крім того, полегшує масштабування таблиць.

Змінено поведінку ряду операторів для забезпечення більшої сумісності зі стандартом SQL:2003.

Порядкова реплікація (row-based реплікація), при якій в бінарний лог буде записуватись тільки інформація про реально змінені рядки таблиці замість оригінального (і, можливо, більш повільного) тексту запиту.

Вбудований планувальник робіт, що періодично запускаються. По синтаксису додання задачі схоже на додання тригера до таблиці; по ідеології — на crontab. Додатковий набір функцій для обробки XML, реалізація підтримки XPath. Нові засоби діагностики проблем і утиліти для аналізу продуктивності. Розширено можливості з керування вмістом лог-файлів, логи тепер можуть бути збережені і в таблицях general_log і в slow_log. Утиліта mysqlslap дозволяє провести тестування навантаження БД із записом часу реакції на кожний запит.

Для спрощення операції оновлення підготовлена утиліта mysql_upgrade, яка виконає перевірку всіх існуючих таблиць на предмет сумісності з новою версією, і при необхідності виконає належні коригування.

MySQL Cluster тепер йде як окремий продукт, який базується на MySQL 5.1 і сховищі NDBCLUSTER. Значні зміни в роботі MySQL Cluster, такі, як, наприклад, можливість зберігання табличних даних на диску.

Повернення до використання вбудованої бібліотеки libmysqld, відсутньої в MySQL 5.0.

API для плагінів, що дозволяє завантажувати сторонні модулі, які розширюють функціональність (наприклад, повнотекстовий пошук), без перезапуску сервера. Реалізація парсера повнотекстового пошуку у вигляді plugin. Новий рушій таблиць Maria (стійкий до збоїв клон MyISAM), який у 2010 був перейменований на Aria та став основою форку MySQL від Монті Віденіуса під назвою MariaDB [14].

2 СПЕЦІАЛЬНА ЧАСТИНА

2.1 Загальна структура системи

Система включає в себе API, та Android додаток. API поділена на декілька ролей:

1. Для «Диспетчера прийому» (медпрацівник, який приймає дзвінки), від громадян. (API інтегрований с системою IP телефонії Asterisk).

2. Для «Диспетчера напрямку» (медпрацівник, який дивлячись на карту, де яка машина знаходиться, та її статус «Є декілька статусів», направляє на машину швидкої медичної допомоги дані (Форма 109/о) звідки їм надійшов виклик «Прізвище, ім’я, вік, код МКХ-10, адресу, контактні номера, номер під’їзду, пароль від домофону, та приблизний опис проблеми, яку диспетчери прийому отримали с телефонної розмови.») — ці всі данні приходять на планшет “Android додаток”.

3. Для «Головного Фельдшера» особа яка назначає зміни працівникам медичної допомоги, тим самим даючи їм змогу авторизуватися в системі. (Якщо медпрацівник не знаходиться на зміні, він не має доступу до системи).

4. «Голова відділу» (медпрацівник, «головний в ШМД» який має змогу прослуховувати дзвінки громадян до ШМД, дивитись статистику викликів, «Які надійшли до ШВД, кількість прийнятих, пропущених (перенаправленних до інших відділів ШМД), кількість надання консультацій, кількість виїздів та в які саме райони, тощо»).

5. «Роль для медпрацівників які споряджають машини ШМД лікарськими засобами» – (Бригада ШМД коли приїжджає на виклик до хворого, заповнює (Форму 110), в якій вказуються всі симптоми та стан пацієнта, а також що вони використали для покращення його стану, або реанімації (перчатки, таблетки, маски, шприци, ампули, тощо…), це дозволяє працівникам розуміти що потрібно додати в комплектацію, та підраховувати витрати).

2.2 Загальна структура бази даних

Для вибору системи управління базою даних (СУБД), яка буде використовуватися прикладним програмним інтерфейсом, варто вдатися до розгляду основних програмних продуктів, які затребувані в даний момент часу на ринку програмного забезпечення, вони резюмовані в таблиці 1(табл. 2.1).

Таблиця 2.1 — Переваги та область застосування сучасних СУБД

Назва	**Переваги**	**Область застосування**
Microsoft™ SQL Server	1. Масштабованість і надійність; 2. Можливість одночасної роботи декількох користувачів з даними; 3. Забезпечення захисту даних у разі збою; 4. Можливість роботи з великими обсягами даних; 5. Можливість аналітичної обробки інформації, що зберігається.	Використовується для роботи з базами даних розміром від персональних до великих баз даних масштабу підприємства
Microsoft™ Access	1. Легкість в освоєнні; 2. Потужні засоби підготовки звітів; 3. Швидке і легке створення баз даних (присутній конструктор, немає необхідності використовувати при створенні БД запити); 4. Використання засобів автоматизації і додавання складних виразів без написання коду; 5. Невимогливий до ресурсів комп'ютера.	Створення звітів довільної форми на підставі різних даних. Розробка некомерційних додатків.

Oracle™ Database	1. Безліч підтримуваних платформ; 2. Швидкість роботи; 3. Легкість в управлінні; 4. Масштабованість; 5. Високий рівень захисту даних (як від апаратних збоїв, так і від атак ззовні); 6. Дозволяє оперувати величезними обсягами інформації.	Бази даних з великим об'ємом інформації.

Була обрана система управління базою даних «Navicat Premium 15» (рис. 2.7). Navicat Premium це дуже потужний інструмент для адміністрування баз даних, який дозволяє підключатися до MySQL, SQLite, Oracle і PostgreSQL баз даних одночасно в одному додатку, що робить адміністрування декількох видів баз дуже простим і зручним.

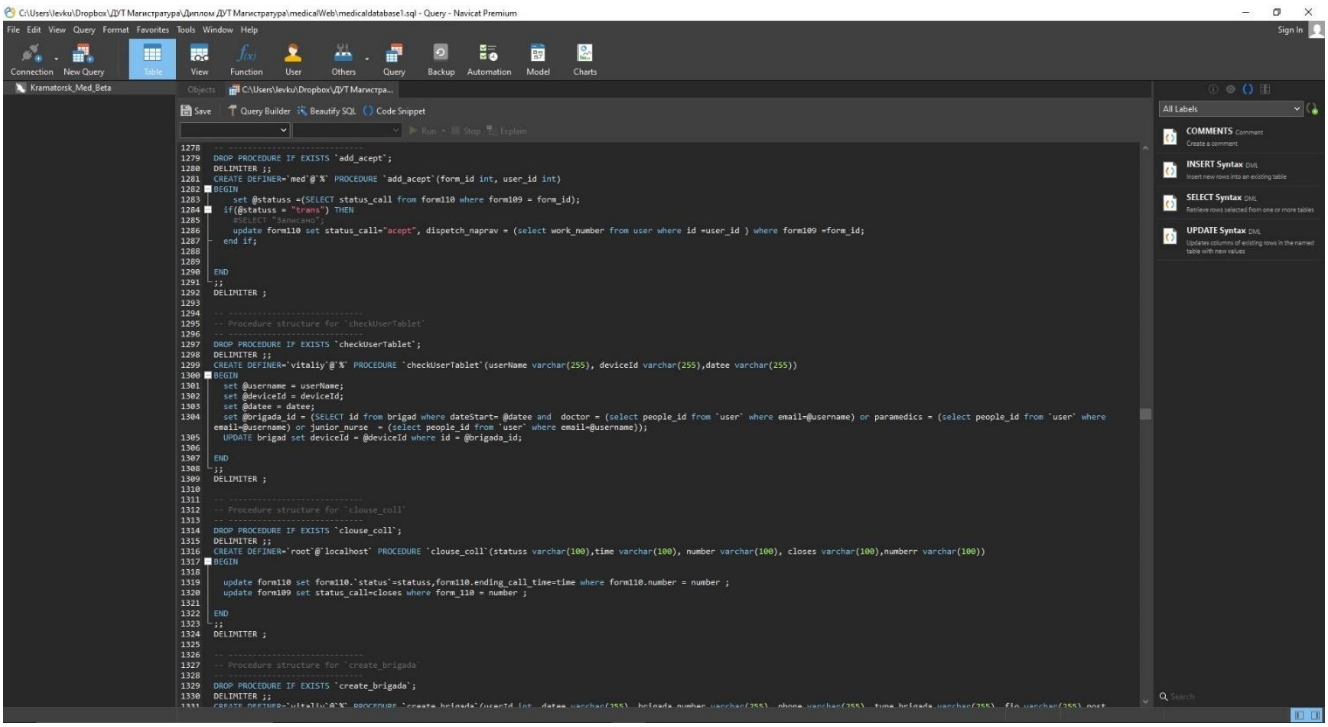


Рисунок 2.7 — Система управління базою даних Navicat Premium

Navicat Premium поєднує в собі функції інших додатків Navicat. При наявності сполук, створених для різних типів баз даних, він дає змогу легко переносити дані між базами MySQL, PostgreSQL і Oracle. Він підтримує більшість функцій в MySQL, SQLite, Oracle і PostgreSQL, включаючи процедури збереження, події, тригери, функції, перегляд і т.д.

Пакетна обробка для різних видів баз даних може бути також запрограмована і запущена в певний час. Інші функції включають майстер імпорту / експорту, майстер складання запитів, майстер звітів, синхронізацію даних, резервне копіювання, планувальник завдань і багато іншого. Можливості Navicat досить великі, щоб забезпечити професійним розробникам універсальну утиліту для їх роботи, але в той же час вони легкі для вивчення користувачами, які є новачками в роботі з серверами баз даних.

Основні функції Navicat Premium:

- HTTP Tunnel;
- SSH Tunnel;
- Синхронізація даних і структури;
- SQL консоль;
- Підтримка множинних з'єднань для локальних і віддалених серверів БД;
- Створення і видалення баз даних, таблиць, індексів і користувачів;
- Підтримка Unicode;
- Імпорт / експорт даних в 5 найбільш популярних форматів: XLS, CSV, TXT, DBF and XML;
- Створення і запуск SQL queries;
- Можливість виконання основних завдань за розкладом;
- Підтримка перенесення даних з одного сервера БД на інший;
- Бекап і відновлення баз даних;
- Управління правами доступу [Security Configuration];
- ER-модель даних;
- Побудова моделі даних;

- SQL Minifier;
- Широкий пошук по БД;
- Можливість запуску операції резервного копіювання для баз даних MySQL (InnoDB only);
- Доданий список об'єктів в редакторі запитів;
- Додана DDL вкладка, Інформація про об'єкт [20].

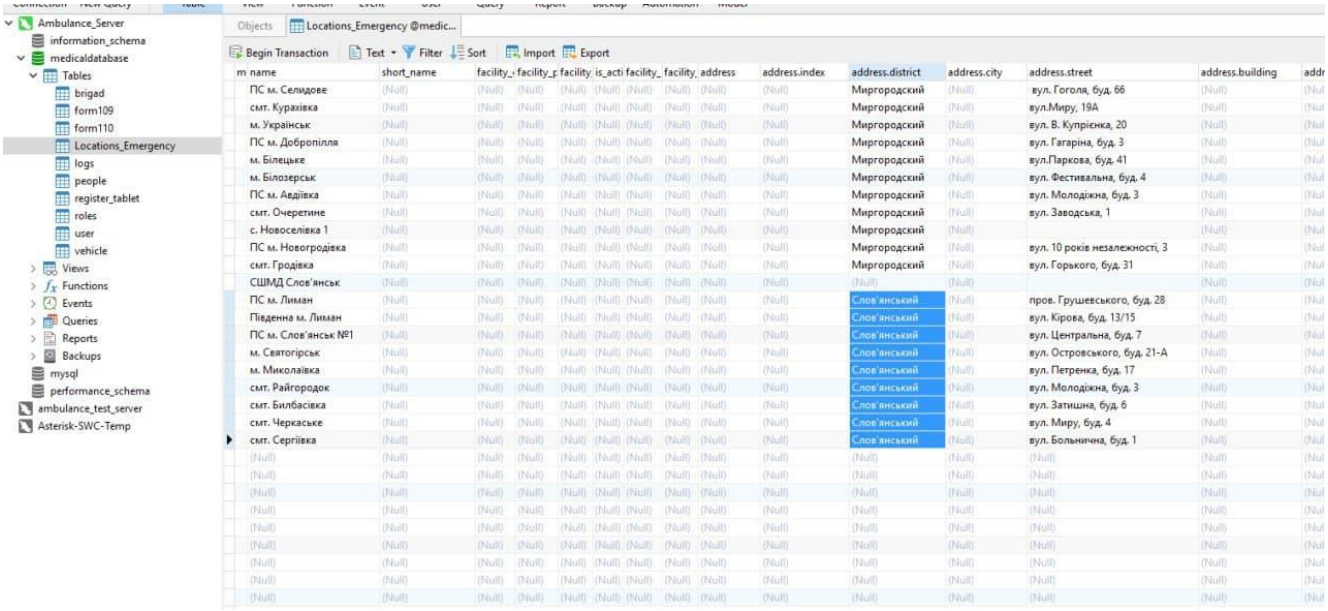


Рисунок 2.8 — База даних «medicaldatabase»

База «medicaldatabase» складається з декількох таблиць (brigad, form109, form110, Locations_Emergency, logs, people, register_tablet, roles, user, vehicle) таблиці являються основою роботи API (рис. 2.8).

Для організації медичних працівників використовується таблиця «user», в неї вносяться майже всі дані про працівника (рис. 2.9).

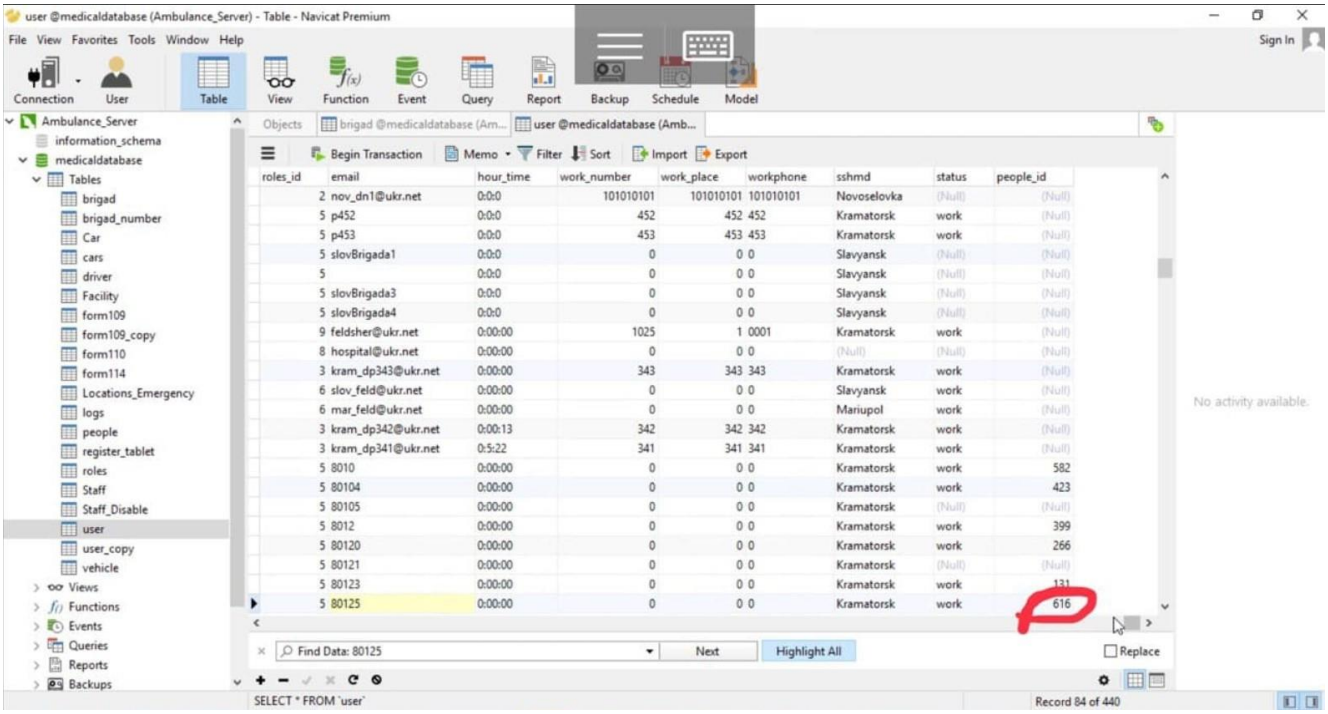


Рисунок 2.9 — База даних «medicaldatabase»

2.3 Технічне завдання для реалізації диспетчера прийому

1. Прийняття виклику через систему. Визначення телефону абонента.

2. Заповнення електронної карти Ф 109/о та кодування електронної карти (визначення режиму реагування).В окремому віконці – інформація для диспетчера з переліком кодів [7].

3. При заповненні електронної карти Ф109/о, якщо виклик вже коли-небудь надходив з даного телефону, з даної адреси або до даного пацієнта, то в окремому віконці відображається інформація про всі виклики, які приймалися з цього номеру телефону або на цю адресу, або з таким же прізвищем, з вказівкою дати та часу, і диспетчер для економії часу встановлює це віконце у нову картку виклику з можливістю корегування та доповнювання.

4. Відправлення карти Ф 109/о диспетчеру напрямку (табл. 2.2 – карта виклику Ф 109/о).

Таблиця 2.2 — Карта виклику Ф 109/о

	Ідентифікаційний код за ЄДРПОУ	
Міністерство охорони здоров'я України Найменування закладу _____ _____		Медична облікова документація Форма N 109/о Наказ МОЗ 17.11.2010 N 999

ЧАС	Прийому: години ____ хвилини ____ Передачі: години ____ хвилини ____ Виїзду: години _____ хвилини _____ Приїзду: години ____ хвилини _____ Закінчення: години ____ хвилини ___ Повернення на підстанцію: години ____ хвилини _____	"___" _____ 20__ року N _____ Диспетчер N_____ Робоче місце N _____ Привід до виклику: _____
ПЕРЕДАНО	підстанції N _____ бригаді ШМД N _____ у поліклініку, пункт невідкладної медичної допомоги, інше (вказати) _____ тел. _____ реєстр. N _____ Прийняв: _____ Передав: _____	Адреса виклику: Район (масив) _____ вул. _____ буд. _____ корп. _____ кв. ____ під'їзд _____ поверх ____ Вхід з: вулиці двору тел. _____ Прізвище _____ Вік _____ Стать Ч/Ж Хто викликає _____
Диспетчер напрямку N_____	Додаткові відомості, причина відмови: _____ _____ _____	АЗС: Час виїзду / Час повер-нення; Довідки про стан виконання виклику: Час / Диспетчер N___

5. Інформування диспетчера про виклики, які приймаються в даний час (наявність віконця у кутку монітору: наприклад, «Зараз приймається виклик адреса, М65, погано гіпертоніку, індивідуальним номером диспетчера який приймає виклик. Дані у віконці заповнюються під час набору даних диспетчером, який приймає виклик).

6. Наявність мапи з можливістю визначення місця адреси виклику та розташування бригади, якій було передано цей виклик (при натискання на значок адреси відображається адреса виклику, номер бригади, якій надано цей виклик, час надходження та передачі виклику (табл. 2.3 та табл. 2.4).

Таблиця 2.3 — Звіт по виклику

Звіт по виклику

№ карти виклику:	3560
№ карти виїзду:	3530
Обробка виклику	Звіт "Час"
Дата:	07.06.2018
Результат виклику:	обслужений
ПІБ:	Левченко Максим Євгенович
Адреса:	м. Краматорськ, вул. Н. Курченко, б. 4, кв. 2
Телефон:	099-282-33-00
Діагноз:	Гіпертонічна хвороба. Криз неускладнений.

Таблиця 2.4 — Звіт по виклику «Час»

Звіт по виклику "Час"

	Кому	**Час**	**Відповідальний**	**Час обробки**	**Сумарний час**
Надійшов:	ЦОД	7:41:43	Вороніна А.А.		
Переданий	ОДВ СШМД Краматорськ	7:42:49	Петренко С.С.	0:01:06	0:01:06
Переданий	бр. № 403	7:42:56	Петренко С.С.	0:00:07	0:01:13
Знятий з бригади		7:43:10	Петренко С.С.	0:00:14	0:01:27
Переданий	бр. № 405	7:43:12	Петренко С.С.	0:00:02	0:01:29

Час прибуття на виклик	бр. № 405	7:49:00		0:05:48	0:07:17

При натисканні на значок бригади – інформація про номер бригади, статус бригади, ПІБ лікаря, скільки викликів вже обслуговано бригадою, середній час обслуговування виклику, номер виклику та адресу, куди їде бригада, час надходження та час передачі виклику, скільки часу бригада обслуговує виклик з моменту отримання виклику (рис. 2.10).

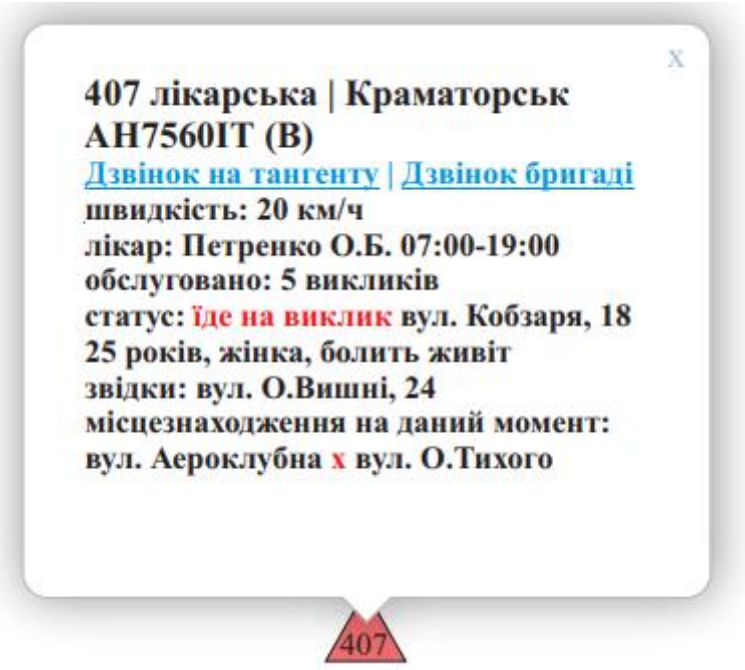


Рисунок 2.10 — Інформація про бригаду

7. Наявність вікон з інформацією про передані та не передані (активні) виклики, з позначкою у не переданих (активних) викликах режиму реагування (екстрені або не екстрені), а у переданих викликах – номеру бригади, яка обслуговує даний виклик, часу надходження виклику та часу передачі виклику бригаді та часу закінчення обслуговування виклику (рис. 2.11 та рис. 2.12).

Інформація про бригади на даний момент часу при НС

- Всі СШМД ☐
- СШМД м. Донецьк ☐
- СШМД м. Волноваха ☐
- СШМД м. Костянтинівка ☑
- СШМД м. Краматорськ ☑
- СШМД м. Маріуполь ☐
- СШМД м. Мирноград ☐
- СШМД м. Слов'янськ ☑

Адреса виклику	Вільні	Відстань до адреси виклику, км	Орієнтовний час доїзду до адреси виклику	На виклику (з зазначенням часу в продовж якого бригада зайнята)
	301-фельдшерська	3.300	0:02:05	302-фельдшерська (00:46:11)
	412 - фельдшерська	4.630	0:02:33	305-фельдшерська (00:33:00)
	303-лікарська	5.954	0:02:57	311-лікарська (01:01:01)
	304-фельдшерська	6.300	0:03:14	
	703 - лікарська	7.877	0:05:46	

Рисунок 2.11 — Інформація про бригади на даний момент часу

Показники роботи бригади за часом

№ з/п	СШМД	ПС	ППБ	№ бригади	Затримка виїздів більше 2 хв.	Перебування на виклику більше 2 хв.	Перебування в прийомному відділенні ЛПЗ більше 15 хв.

Рисунок 2.12 — Показники роботи бригади за часом

8. Наявність швидкого зв'язку із службою порятунку та іншими екстреними службами.

9. Наявність зворотного зв'язку з абонентом.

10. Надання кожному диспетчеру індивідуального номеру за наступним принципом:

1. ОДВ СШМД м. Донецьк – Д101, Д102,……Д114;

2. ОДВ СШМД м. Волноваха - Д201, Д202,…..Д214;

3. ОДВ м. Костянтинівка - Д301, Д302,…..Д320;

Для диспетчерів резерву надання логіну та паролю для кожного з літерою R (резерв) в позначці оператора. Наприклад: В. Новосілка – Оператор: R1 Іваненко О.О.., Волноваха: - Оператор: R2 Петренко Л.Л., Костянтинівка –

Оператор: R3 Голубенко Р.Р., Краматорськ – Оператор: R4 Сидоренко С.С., Маріуполь – Оператор: R5 Горбенко К.К., Мирноград – Оператор: R6 Вовченко З.З., Слов'янськ – Оператор: R7 Вишня Г.Г.

11. Наявність інформації про всі виклики до кожного конкретно хворого за будь-який термін часу.

13. Наявність панелі з довідковою інформацією про коди скарг, ключові питання диспетчера для бесіди з абонентом та порадами для надання до медичної допомоги до приїзду бригади ЕМД (після диспетчерська підтримка).

14. Доступ до архіву звернень.

2.4 Технічне завдання для реалізації диспетчера направлення

1. Наявність карти з зображенням бригад, які працюють у дану зміну (рис. 2.10), з позначкою їх номеру, статусу (зелений - вільна, червоний – їду на виклик, рожевий - на виклику, жовтий - транспортування, фіолетовий – тех. сервіс), типу бригади (лікарська - у формі трикутника, фельдшерська - у формі кола, психіатрична - у формі квадрату). При натисканні на позначку бригади повинна з'являтися піктограма з наступною інформацією: номер бригади, тип, належність до СШМД чи ПБ, ПІБ старшого бригади, час початку та закінчення зміни, кількість обслугованих викликів, номер карти виїзду, адреса виклику, привід виклику, час надходження, передачі цього виклику, час обслуговування цього виклику, Якщо бригада рухається - відображення її місцеположення та швидкості руху.

2. Радіус визначення вільних бригад у ОДВ СШМД повинен відповідати розмірам зони обслуговування станції (у ОДВ Центру – повинна бути можливість визначати вільні бригади вручну та з наявністю радіусу).

3. Виклики, які надходять від диспетчера прийому до диспетчера направлення повинні бути розділені за режимом реагування на CITO (нещасний випадок, (ДТП, ЧП) - червоний колір, ЕКСТРЕНІ - жовтий, НЕЕКСТРЕНІ - зелений.

4. В окремому вікні диспетчера направлення повинні бути колонки з переданими викликами та викликами, які потрібно передати (рис. 2.11).

5. При натисканні на виклик, який обслуговує бригада, на карті повинні відображатися місцеположення адреси з піктограмою. На піктограмі відображається наступна інформація: адреса виклику, стать, вік пацієнта, привід виклику, номер бригади, яка обслуговує виклик, час надходження та передачі виклику (табл. 2.2, та табл. 2.3).

6. Повинно бути вікно з переліком всіх бригад, які працюють у дану зміну (рис.2.13 та рис. 2.11).

Вікно з переліком бригад які працюють в дану зміну								
Дата	**СШМД : ПС, ПБ**	**№ Бригади**	**ПІБ та посада кожного члена бригади**	**Час початку та закінчення зміни**	**Кількість обслуго ваних викликів**	**Вільний час з останнього виклику**	**Місце розташування бригади на даний момент**	**Подальші дії бригади**
			Лікар: ПІБ	07:00-07:00			1) якщо на виклику (адреса виклику, № талона виїзду, час отримання виклику; 2) якщо вільна-де вона зараз знаходиться?	
			Фельдшер: ПІБ	07:00-19:00				
			Санітар: ПІБ	07:00-07:00				
			Водій: ПІБ	07:00-19:00				

Рисунок 2.13 — Показники роботи бригади за часом

При натисканні на позначку будь-якої бригади повинна надаватись інформація наступного характеру: номер бригади, ПІБ мед. працівника, початок та закінчення зміни, кількість обслугованих викликів, зазначення вільного часу з останнього виклику та місце розташування на даний момент, а якщо бригада обслуговує виклик, то адресу виклику, номер талону виїзду, привід виклику, час обслуговування виклику, подальші дії бригади (транспортування хворого, залишення на місці, відмова від допомоги, хибний виклик тощо) (рис. 2.10).

7. Передбачити інформування диспетчера сигналом про затримку виїзду бригади на виклик більше 2 хв., перебування в приймальному відділенні ЛПЗ більше 15 хв., знаходження бригади на виклику більше 1 год.

8. На мапі повинна бути інформація про кількість працюючих в дану зміну машин, кількість машин, зв'язок з трекерами яких раптово зник, кількість машин

з трекерами, які рухаються, але не працюють в дану зміну (таблиця №6).

9. Наявність швидкого зв’язку із службою порятунку та іншими екстреними службами.

10. Наявність зворотного зв’язку з абонентом.

14. Доступ до архіватора.

15. Диспетчер направлення перед початком зміни формує список бригад, які будуть працювати в дану зміну (рис. 2.14).

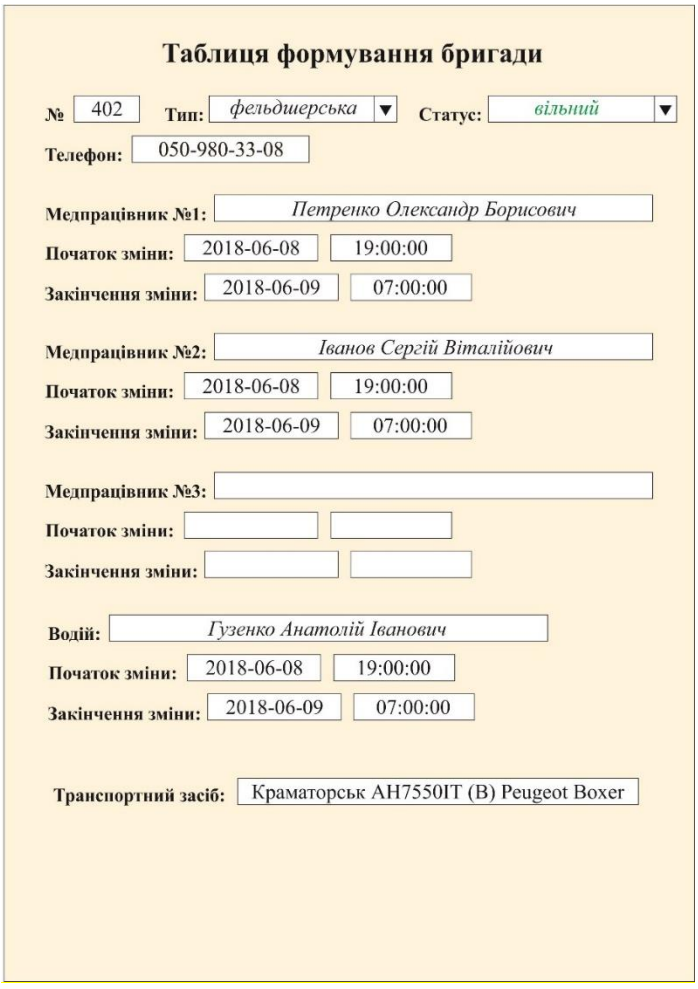


Рисунок 2.14 — Формування бригад

16. Додати диспетчеру напрямку графік етапного часу (рис. 2.15).

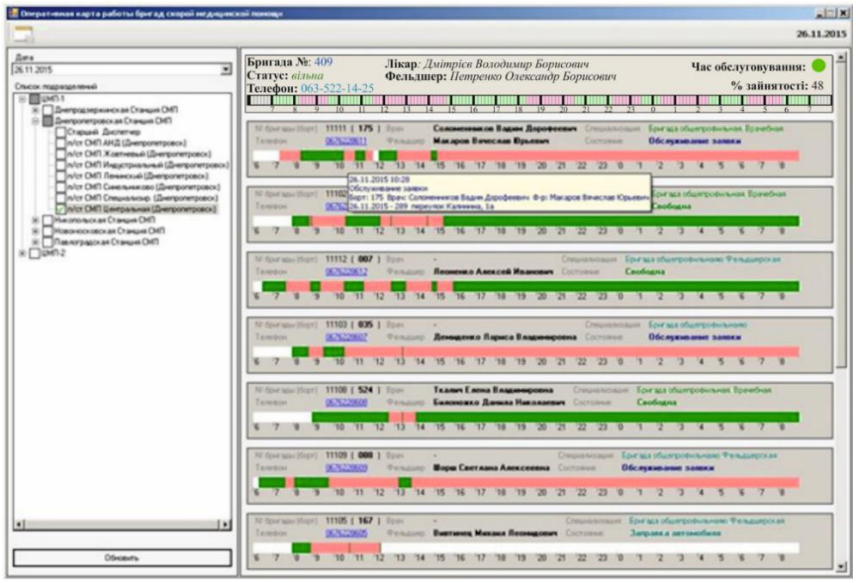


Рисунок 2.15 — Графік етапного часу

При натисканні на номер бригади повинно вискакувати віконце з інформацією наступного характеру:

- ПІБ кожного члена бригади (у тому числі і водія) з відміткою часу спочатку та закінчення зміни;
- Номер автомобілю;
- Кількість обслугованих викликів на момент спостереження;
- Адреса знаходження бригади на момент спостереження.

При натисканні номера телефону повинно відбуватися з'єднання з бригадою.

Час обслуговування:

- Якщо час обслуговування виклику відповідає межам нормативного часу, індикатор зостається білим, або прозорим (не зеленим, як на малюнку - ми помилилися);
- Якщо час обслуговування перевищує одну годину - індикатор чорніє та починає блимати.

Процент зайнятості (по часу) для кожної бригади на протязі зміни: система повинна обчислювати сумарний час обслуговуваних викликів з початку зміни до моменту спостереження (для наглядності навантаження) у відсотках.

2.5 Технічне завдання для реалізації дій працівника бригади ШМД

Отримавши виклик на планшет, працівник мобільної бригади натисканням клавіші сигналізує диспетчеру напрямку про прийом виклику. При цьому на мапі монітора диспетчера напрямку значок бригади з зеленого кольору змінюється на червоний (статус «зайнята, їду на виклик»). Бригада їде на виклик, а на планшет мобільної бригади приходить інформація про пацієнта, до якого їдуть (всі діагнози, скільки разів викликав, де і яку медичну допомогу отримував і т.п.).

По прибуттю на виклик мобільна натискає клавішу і диспетчер напрямку отримує інформацію про те, що бригада прибула на місце виклику (при цьому на моніторі диспетчера напрямку значок бригади змінює свій колір з червоного на рожевий - статус «чекаю бригаду»).

При обслуговуванні виклику мобільна бригада заповнює паспортну частину карти виїзду (форма 110/о) (рис. 2.16 та рис. 2.17), надає медичну допомогу. Якщо хворий підлягає госпіталізації, мобільна бригада повідомляє по телефону про це диспетчера напрямку. Ця інформація потрібна в разі, якщо стан пацієнта критичний і медпрацівники ЛПУ повинні прийняти його якомога швидше (диспетчер напрямку повинен повідомити про це на санпропускник лікарні) .При транспортуванні пацієнта мобільна бригада знову змінює статус бригади - «везу пацієнта», значок бригади змінюється на жовтий [8].

СТАНЦІЯ ШВИДКОЇ МЕДИЧНОЇ ДОПОМОГИ | форма N 110/о, Наказ МОЗ 17.11.2010 N 999

КАРТА ВИЇЗДУ ШВИДКОЇ МЕДИЧНОЇ ДОПОМОГИ N ____ А ____ 0 ____

Адреса виклику	
Привід до виклику	**Район**

Прізвище (зі слів)	Друкованими літерами	
Ім'я (з документа)		
По батькові		
Адреса особи	**Район**	
Місце роботи	**Посада**	
Страховий поліс N	**Страхова компанія**	

1 **Вік** дні ____ міс. ____ роки ____

2 **Стать:** 1 - чоловік; 2 - жінка

3 **Мешканець:** 1 - міста; 2 - села; 3 - іншої держави

4 **Соціальний стан:** 1 - робітник; 2 - службовець; 3 - учень; студент; курсант; 4 - підприємець; 5 - пенсіонер; 6 - інвалід; 7 - утриманець; 8 - фермер; 9 - не працює; 10 - біженець; 11 - без ПМП

5 Подальше перебування особи:
1 - на місці; 2- доставлений додому; 3- в інше місце (вказати) _____
4 - за направленням ШМД N наряду _____
5 - за направленням інших ЛПЗ _____
6 - доставлений в травмпункт, поліклініку
N супровідного листа _____

У лікарню N _____ відділення _____
Час з _____ по _____ ; прийнятий, неприйнятий
Діагноз _____
Черговий лікар _____

У лікарню N _____ відділення _____
Час з _____ по _____ ; прийнятий, неприйнятий
Діагноз _____
Черговий лікар _____

6

Повідомлення:	**Телефон**	**Час**	**Прийняв**
Ст. лікарю ШМД			
Поліклініку(ПНМД)			
СЕС			
РВВС			
ДАІ			
Комендатуру			
Родичам			
Стіл довідок ШМД:			
Інше місце			

З пацієнтом передано:
_____
_____
_____
_____
_____
Здав: **Прийняв:**

Склад бригади:
Лікар _____
Фельдшер _____
Мол. м/сестра _____
Водій _____
Стажер _____

Супроводжуючий

7 **Особистий N медпрацівника** ____

8 **Код підстанції** ____

9 **Код бригади ШМД** ____

10 **Допомога за діагнозом:** 1 - термінова; 2 - нетермінова

Час

11	**Прийому виклику**	____
12	**Виїзду на виклик п/с, р/с, Т**	____
13	**Прибуття на місце**	____
13.1	**Виїзду на госпіталізацію (інше)**	____
14	**Закінчення виклику**	____
15	**Повернення на станцію**	____
	Кілометраж	____

16 **Тип виїзду за діагнозом:** 1 - нещасний випадок; 2 - раптове захворювання; 3 - пологи; 4 - патологія вагітності; 5 - перевезення хворого; 6 - хронічне захворювання; 7 - технологічний виїзд; 8 - чергування; 9 - безрезультатний ____

17 **Обгрунтованість:** 1 - профільний; 2 - непрофільний ____

18 **Місце виклику:** 1 - ЛПЗ; 2 - квартира; 3 - інше (вказати) _____; 4 - громадське місце; 5 - робоче місце; 6 - вулиця; 7 - навчальний заклад ____

19 **Виклик** 1 - первинний; 2 - повторний; 3 - попутний; 4 - амбулаторний ____

20 **Звернення хворого:** 1 - не звертався; 2 - звертався на СШМД (скільки разів ___); 3 - звертався в інший ЛПЗ ____

21 **Травма:** 0 - немає; 1 - побутова; 2 - вулична; 3 - кримінальна; 4 - ДТП; 5 - виробнича; 6 - спортивна; 7 - самогубство; 8 - вулична, внаслідок ожеледиці ____

22 **Час від початку хвороби:** 1 - до 1 год; 2 - від 1 до 3 год; 3 - від 3 до 6 год; 4 - від 6 до 12 год; 5 - від 12 до 24; 6 - понад 24 год ____

23 **Алкоголь:** (зі слів) 1 - вживав; 2 - не вживав;3 - не відомо ____

24 **Результат:** 1 - покращення; 2 - без ефекту; 3 - погіршення; 4 - смерть в присутності; 5 - смерть до приїзду; 6 - викликана спецбригада; 7 - здоровий; 8 - чергування; 9 - інші перевезення; 10 - заправка автомобіля; 11 - інше (вказати) _____ ____

25 **Безрезультатний виїзд:** 1 - не застали; 2 - адреса не знайдена; 3 - не доїхали; 4 - не викликали; 5 - обслужений до приїзду; 6 - відмова від медичної допомоги ____

26 **ПОПЕРЕДНІЙ ДІАГНОЗ:**
_____
_____
_____
_____

Код МКХ - Х ____

Згоден на запропоноване лікування. ***Підпис*** _____

ВІДМОВА

Я,
_____
відмовляюся від / медичної допомоги / транспортування / (підкреслити), які рекомендовані мені медичним персоналом, розуміючи наслідки такої відмови. Підпис особи _____
Свідок _____ Свідок _____

Рисунок 2.16 — Форма 110/о (Сторона 1)

Дата захворювання _____ **Початок захворювання** (Час) _____ год. _____ хв. _____

Скарги:

АНАМНЕЗ ☐ ІХС ☐ ГХ ☐ ГІМ коли _____ ☐ ГПМК коли _____ ☐ Судоми ☐ Діабет ___ тип ☐ ☐ Онко

Алергія ☐ Ні ☐ Так _____ ☐ Акушерсько-гінекологічний _____

☐ Вір. Гепатит ☐ Туберкульоз ☐ Коросга ☐ Педикульоз

ОБ'ЄКТИВНІ ДАНІ **Загальний стан** ☐ Задовільний ☐ Середній ☐ Важкий ☐ Дуже важкий ☐ Термінальний **t° С** ☐

Свідомість ☐ Ясна ☐ Приглушення ☐ Сопор ☐ Кома **за ШГ** ☐ **ТШ** ☐ **Поведінка** ☐ спокійний ☐ збуджений агресивний

Положення ☐ активне ☐ пасивне ☐ вимушене _____ **Зір** _____

Шкіра ☐ звичайна ☐ гіперемія _____ ☐ бліда ☐ іктерична ☐ ціаноз ☐ суха ☐ волога ☐ тепла ☐ холодна

Артеріальний тиск _____ **мм.рт.ст. постійний** _____ **мм.рт.ст. Пульс** _____ хв. Характеристика _____ Дефіцит пульсу _____

Тони серця ☐ ритмічні ☐ аритмічні ☐ ясні ☐ глухі ☐ акцент ____ тону на _____ шум _____ Шоковий індекс ____

Частота дихання ____ хв. **Задишка** ☐ експіраторна ☐ інспіраторна ☐ змішана **Периферичні набряки** _____

Дихання ☐ вільне ☐ поверхневе ☐ термінальне ☐ везикулярне ☐ хрипи ☐ сухі ☐ справа ☐ крепітація _____

☐ утруднене ☐ глибоке ☐ апное ☐ жорстке ☐ послаблене ☐ вологі ☐ зліва ☐ шум тертя плеври _____

Додатково _____

Живіт ☐ участь в акті дихання ☐ м'який ☐ напружений ☐ безболісний ☐ болісний, в ділянці _____

Печінка нижній край щодо реберної дуги ☐ не виступає ☐ виступає на _____ см, _____ **Перистальтика** ☐ так ☐ ні

Симптоми подразнення очеревини ☐ ні ☐ так **Діурез** зі слів _____ **Випорожнення** зі слів _____

Інші симптоми _____

Неврологічний статус Зіниці ☐ норма ☐ міоз ☐ мідріаз ☐ анізокорія D S **Реакція на світло** ☐ так ☐ ні **Очні яблука відведені** _____

Ністагм ☐ ні ☐ так (вказати) _____ **Мова** ☐ збережена ☐ порушена ☐ відсутня **Ковтання** ☐ вільне ☐ порушене

Обличчя ☐ симетричне ☐ асиметричне, носогубна складка згладжена _____ **Девіація язика** _____ **Прикушення язика** ☐ ні ☐ так

Патологічні рефлекси _____ **Тонус м'язів D S** _____ **Менінгеальні ознаки** ☐ ні ☐ так ☐ сумнівні

Плегії, паралічі _____ **Додатково** _____

Місце ушкодження при травмі (обвести номер та описати): **1.** Зовнішня кровотеча (капілярна, венозна, артеріальна) **2.** Опік (термічний, хімічний, електричний) **3.** Електротравма **4.** Ампутація **5.** Вогнепальне поранення **6.** Ножове поранення **7.** Пневмоторакс (відкритий, закритий, напружений) **8.** Тривале стиснення **9.** Падіння з висоти **10.** Травма голови **11.** Травма хребта **12.** Внутрішні пошкодження **13.** Переломи / Вивихи **14.** Пошкодження м'яких тканин **15.** Інше (вказати) _____

Описання ЕКГ, глюкоза крові (ммоль/л) тощо: _____

НАДАНА МЕДИЧНА ДОПОМОГА, ЛІКАРСЬКІ ЗАСОБИ (назва, доза, шлях введення):

☐ **Інгаляція O_2** _____ **л/хв**	
☐ **СЛР** Початок _____	
Закінчення _____	
☐ **ШВЛ** (метод) _____	
☐ **НМС**	
☐ **Інтубація трахеї**	
☐ **ДЕФ** кількість разів _____	
☐ **ЕКС** _____ частота _____	
☐ **Пункція ЦВ** _____	
☐ **Пункція плевральної порожнини**	
☐ **Первинна обробка рани**	
☐ **Зупинка кровотечі** (метод) _____	
☐ **Шийний комір**	
☐ **Іммобілізація кінцівок** _____	
☐ **Довга дошка**	
☐ **Промивання шлунка**	
☐ **Катетеризація сечового міхура**	
☐ **Очисна клізма**	
☐ **Інше** _____	
☐ **В/В** ☐ **В/М** ☐ **ПШ** ☐ **Per / os**	

Стан після надання допомоги:

Загальний стан

АТ _____/_____ мм.рт.ст. **Пульс** _____ хв., _____ **ЧД** _____

за ШГ _____ балів за ТШ _____ балів

Транспортування пацієнта (підфеслити) **пішки на ношах на руках**

Підпис керівника бригади

Рисунок 2.17 — Форма 110/о (Сторона 2)

Якщо медична допомога на виклику надана в повному обсязі, або пацієнт не

бажає їхати в стаціонар, мобільна бригада змінює статус бригади на «вільний», колір значка бригади стає зеленим і бригада готова до прийому чергового виклику.

Якщо на виклику стався напад на бригаду, у мобільної бригади повинна бути можливість за допомогою «тривожної кнопки» повідомити про це диспетчера напрямку та поліцію.

Крім того, диспетчер напрямку повинен отримувати інформацію про перебування на виклику бригади понад 1 годин (миготливе віконце). Остаточне заповнення форми 110/о здійснюється після надання меддопомоги пацієнту; форма передається диспетчеру напрямку (для дозаповнення форми 109/о), а блок інформації з об'ємом наданої допомоги і витраченими медпрепаратами та матеріалами передається на АРМ фельдшера по комплектації бригад (для формування поповнення та видачі його бригаді, коли та повернеться на базу).

При транспортуванні пацієнта в стаціонар мобільна бригада заповнює супровідний лист (форма 114/о), який містить інформацію про паспортні дані хворого, його попередній діагноз і обсяг наданої на догоспітальному етапі медичної допомоги. Частина інформації з форми 114/о заноситься потім в стаціонарі в історію хвороби, а частина (після виписки хворого зі стаціонару) - з уточненими заключним діагнозом - повертається на СШМД. На етапі впровадження медичної інформаційної системи у всіх ЛПУ (вище описаних) форма 114/о буде заповнюватися в електронному вигляді і надходити в санпропускник лікарні під час транспортування хворого.

2.6 Текст програми

Оскільки код API та Android додатку великі за розмірами, деякі файли будуть представлені в додатках до диплома, (Додаток А) та (Додаток Б) відповідно.

3 СТВОРЕННЯ API ТА ANDROID ДОДАТКУ

У даному розділі проводиться аналіз вхідних та вихідних даних, розробляється стратегія створення API та Android додатку для Швидкої медичної допомоги і надається опис реалізації та тестування проекту загалом.

3.1 Аналіз вхідних даних. Стратегія створення API

API «Ambulance» являє собою інформаційну структуру, яка в свою чергу має доступ до мережі Інтернет, а також зв’язана з іншими серверами та окремими базами даних. Перш за все для прикладного програмного інтерфейсу потрібно було створити WEB-інтерфейс «Диспетчера прийому», який задовільнив-би всі потреби замовників та користувачів (рис 3.18).

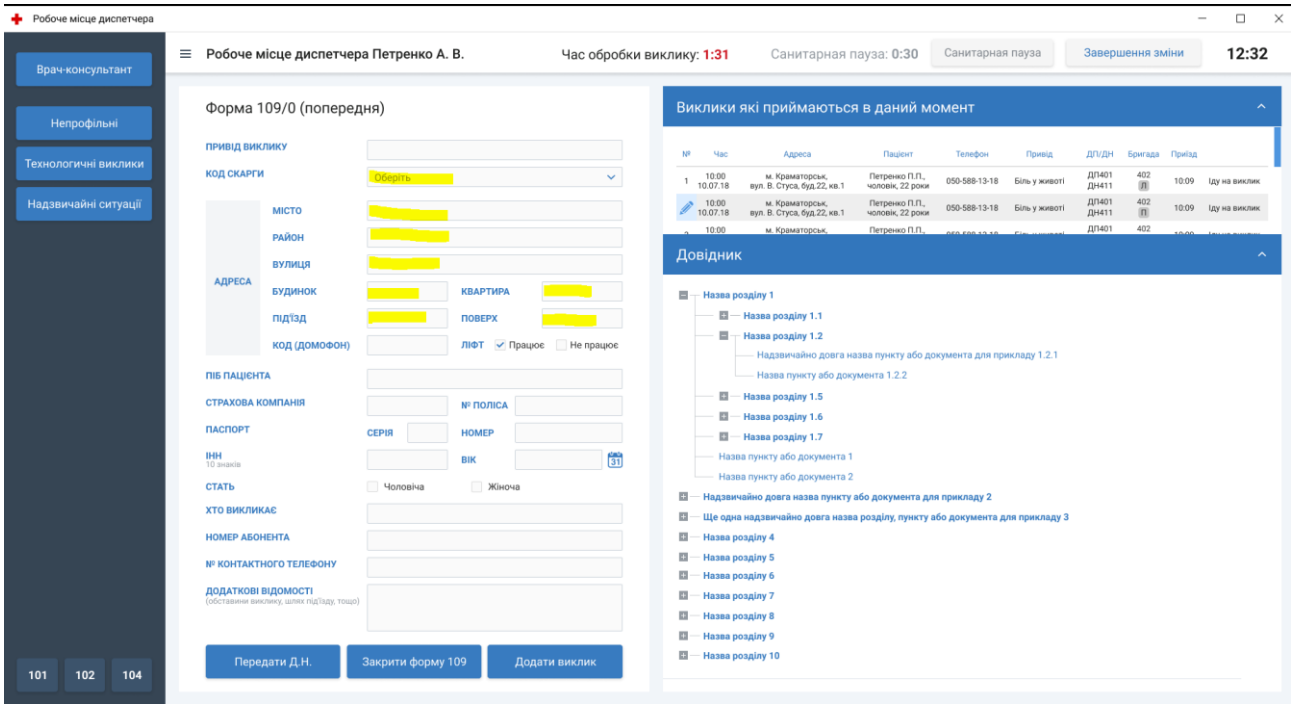


Рисунок 3.18 — Головна сторінка Диспетчера прийому

На головній сторінці веб-інтерфейсу в правому кутку йдуть клавіші швидкого доступу для персоналу (Лікар-консультант, Непрофільні виклики,

Технологічні виклики, надзвичайні ситуації).

Далі йде вікно форми 109/о, кнопки на формі «Передати ДН», «Закрити форму 109/о», «Додати виклик».

1) Модальне вікно «Вхідний дзвінок» з кнопкою «ОК».
2) Модальне вікно «Передано ДН» з кнопкою «ОК».
3) Модальне вікно «Ви впевнені що закрили форму?» з кнопками «Передати ДН» і «Закрити форму 109/о».
4) Модальне вікно «Існуючі зміни» з кнопками «Передати ДН» і «Помилка введення чи передачі даних».
5) Модальне вікно «Непрофільні» з кнопками «Хуліганський», «Відмова у виїзді з обґрунтуванням у формі 109/о», «Передача до ЛПЗ», «Обрив зв'язку».
6) Модальне вікно «Технологічні виклики» з кнопками «АЗС», «Ремонт», «Санітарна обробка», «Допомога в транспортуванні хворого», «Буксирування», «Поповнення медичної сумки», поле «причина».
7) Модальне вікно «Надзвичайних ситуації» з кнопками.
8) Модальне вікно «Консультація диспетчера» з кнопками «Генеральна консультація», «Медична консультація».
9) Модальне вікно «Повторне введення виклику» з кнопками «Передати ДН» і «Помилка введення. Чи не передавати ДН ».
10) Модальне вікно «Повторне введення виклику» з кнопками «Передати ДН» і «Помилка введення. Чи не передавати ДН ».

Алгоритм дій диспетчера прийому викликів (рис. 3.19).

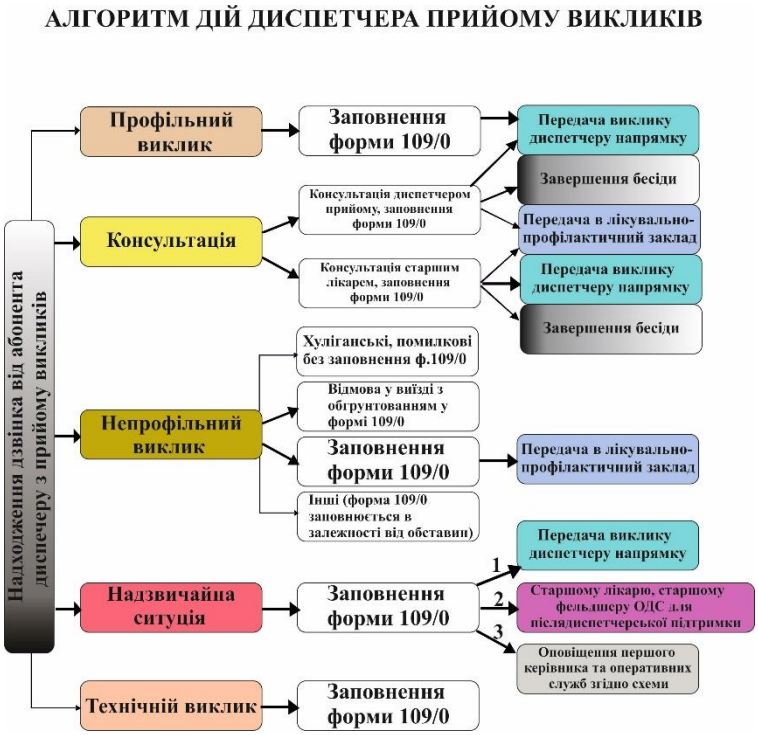


Рисунок 3.19 — Алгоритм дій Диспетчера прийому

Далі потрібно було розробити WEB-інтерфейс «Диспетчера направлення», щоб мати змогу координувати та направляти вільні машини на вказані виклики (рис 3.20). На головній сторінці веб-інтерфейсу в правому кутку йдуть клавіші швидкого доступу для персоналу (Старший лікар-консультант, Довідник телефонів ЛПУ міста).

Функціонал «Диспетчера направлення»:

1) Відправлення певної машини за викликом. Відправлення бригади ШМД може бути:
    a. Відразу (при наявності вільної машини).
    b. За розкладом (санавіація).
    c. Згідно з чергою для даної бригади.
    d. За викликом створеному вручну старшим диспетчером зміни.

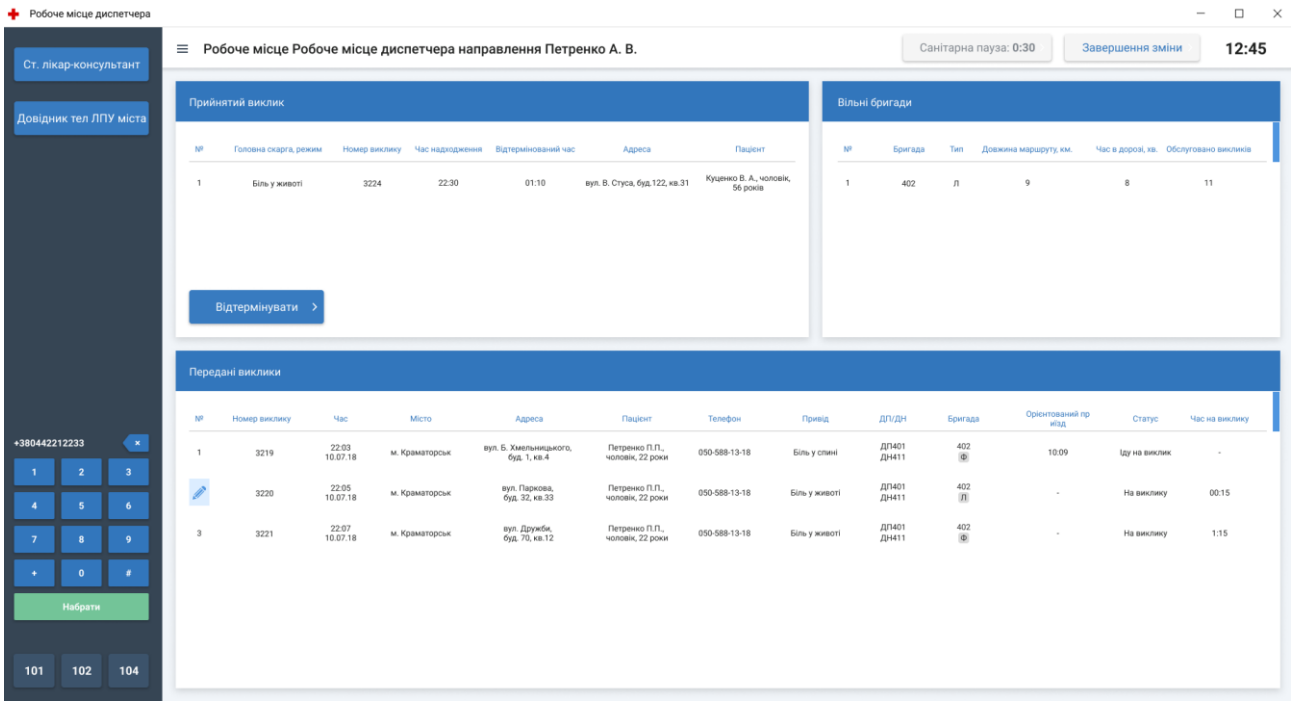


Рисунок 3.20 — Головна сторінка Диспетчера направлення

2) Отримання інформації з будь-яких машин.
3) Отримання інформації про водіїв (ідентифікаційні картки) та інформанії про склад бригади (лікарська або фельдшерська).
4) Доступ до бази даних адрес та машин через API Google Maps.
5) Доступ до прорахунку маршруту руху машини від точки на карті до місця виклику через API Google Maps.
6) Відображення статистики по машині на екрані при кліці на значок машини.

3.2 Аналіз вхідних даних. Стратегія створення Android додатку

Android додаток «Brigada» в свою чергу має доступ до мережі Інтернет, а також зв’язаний зі створеним API «Ambulance». Перш за все для додатку потрібно було створити візуальний інтерфейс «Робоче місце лікаря мобільної бригади», який задовільнив-би всі потреби замовників та користувачів.

При запуску додатку з’являється форма авторизації працівника бригади ШМД. В якій потрібно ввести свій персональний обліковий запис в якості логіну

та пароль (рис 3.21).

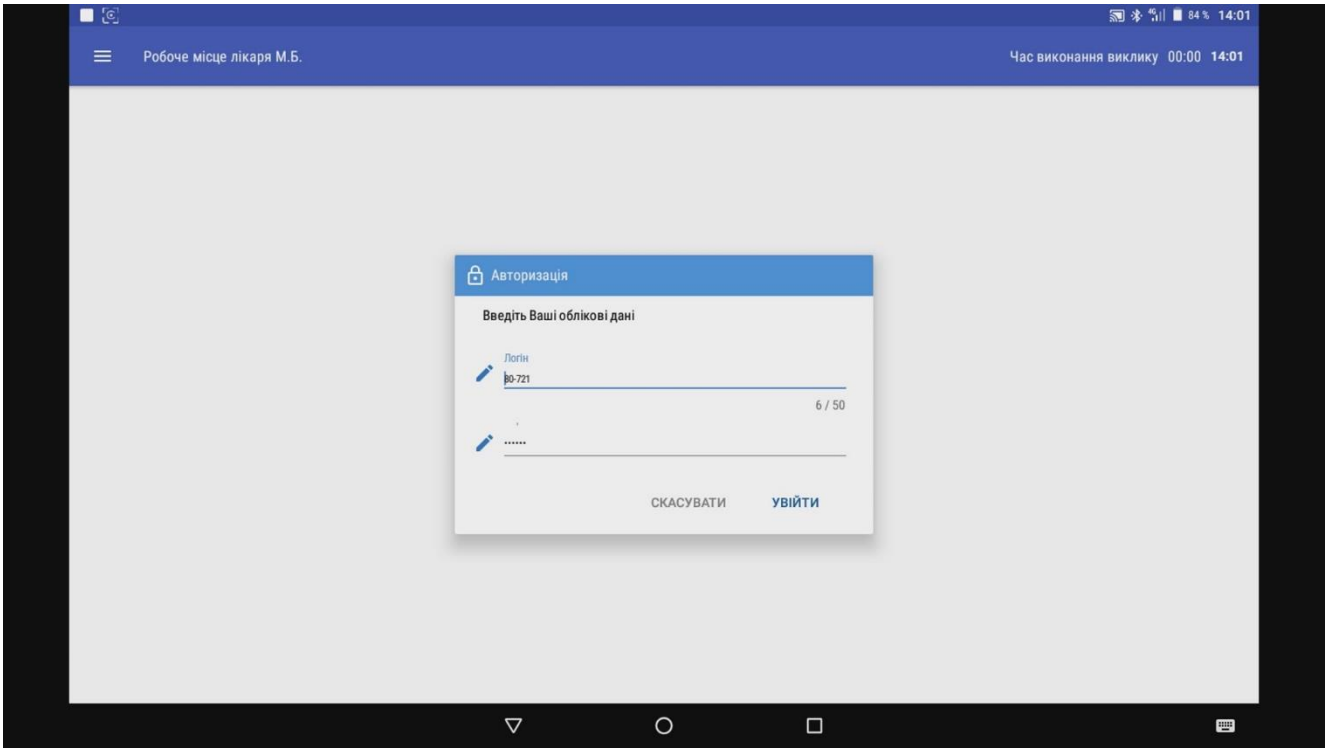


Рисунок 3.21 — Авторизація в Android додатку

Після авторизації працівники мають змогу переглянути на якій машині (№ машини) та тип обладнання (лікарська або фельдшерська) вони будуть працювати, також вони мають змогу переглянути свій графік роботи (Початок зміни та кінець зміни). Переглянути комплектацію сумки з медикаментами, та розпочати роботу (рис 3.22).

Після натиснення кнопки «Почати роботу», працівники мобільної бригади очікують виклик від «Диспетчера направлення». Коли виклик надійшов, на формі 109/о з’являється інформація про пацієнта: Адреса; Код під’їзду; Привід виклику; ПІБ пацієнта; Стать; Вік; Хто викликав швидку; Номер контактного телефону; Додаткові відомості. Після отримання адреси виклику працівники натискають на кнопку «Виїзд», тим самим інформуючи диспетчера направлення що виклик прийнято та машина знаходиться в дорозі.

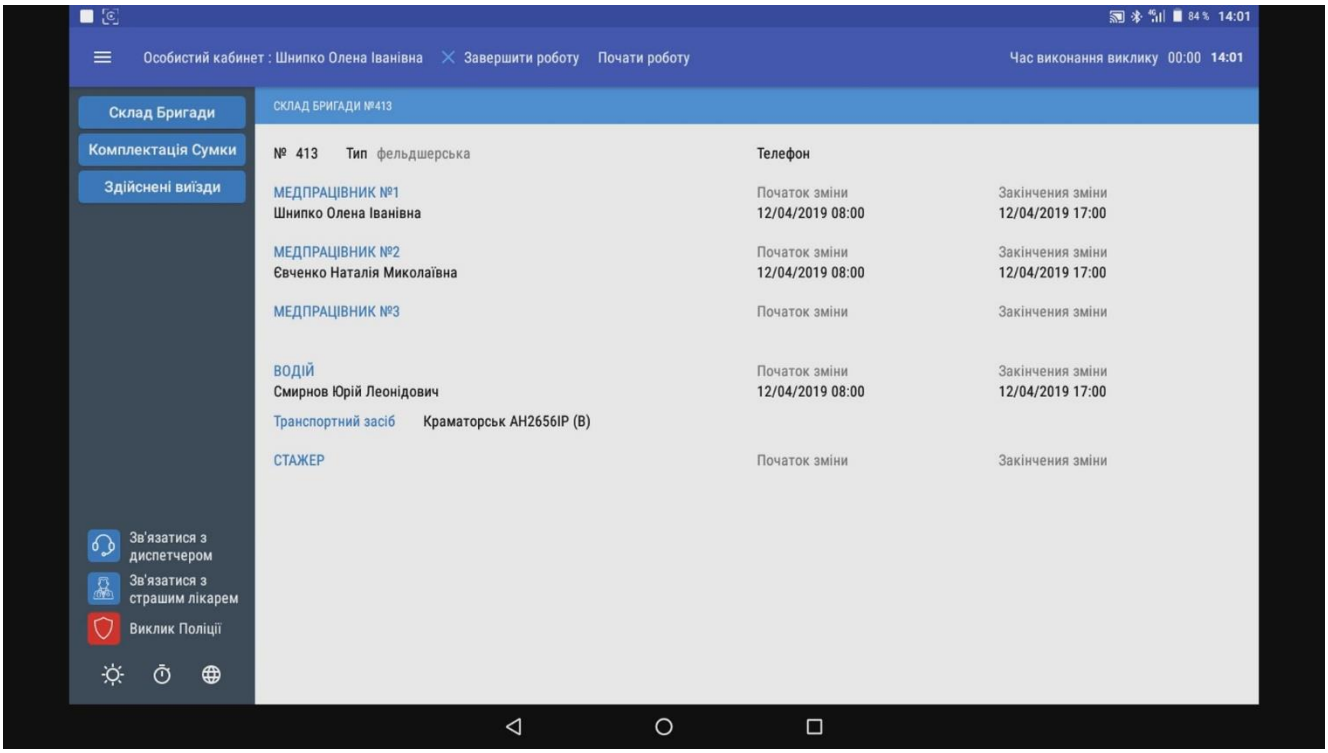


Рисунок 3.22 — Особистий кабінет лікаря мобільної бригади

Не завжди працівники отримують відразу всю інформацію, деякі поля можуть бути пустими, а заповнятися вже на місті виїзду (рис. 3.23).

Після приїзду на місце виклику працівники бригади натискають кнопку «Прибув на місце», інформуючи тим самим диспетчера напрямку про те що мобільна бригада прибула на місце та починає надавати допомогу.

При приїзді на місце виклику та під час надання допомоги працівники відкривають форму 110/о для заповнення даних (рис. 2.16 та рис. 2.17), дана таблиця повністю перенесена в Android додаток. Викликається при натисненні клавіші «Форма 110».

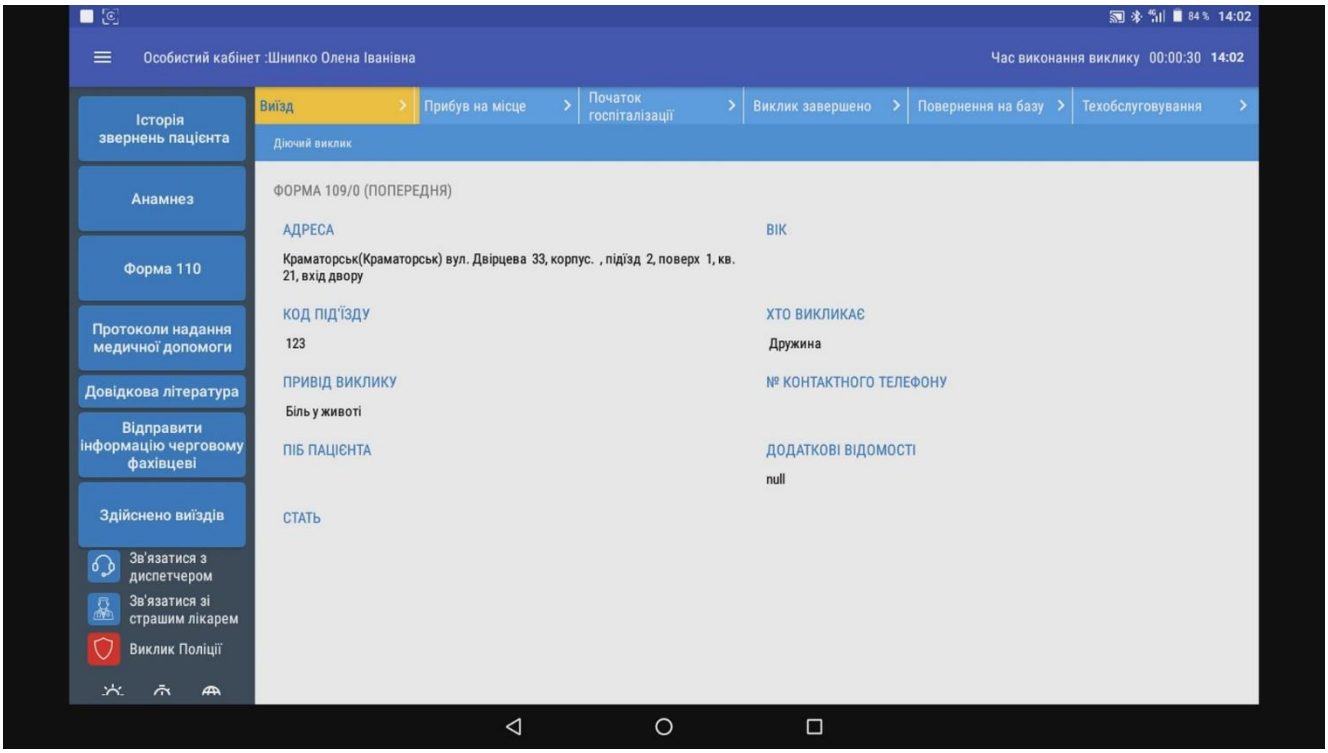


Рисунок 3.23 — Отримання даних на Android додаток

При заповненні форми 110/о попередні дані, які були надіслані диспетчером прийому в форму 109/о автоматично переносяться до форми (Адреса виклику, район, ПІБ, стать, вік), дані можна змінити при необхідності (рис. 3.24).

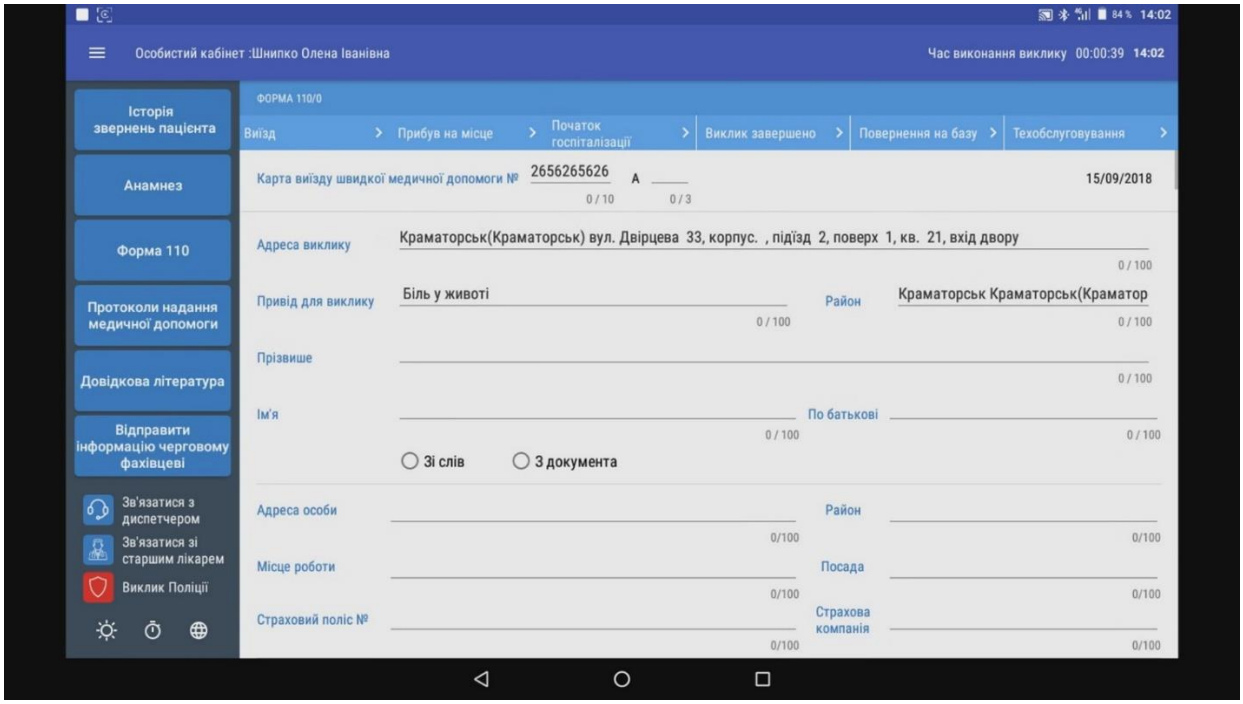


Рисунок 3.24 — Відкриття форми 110/о

Після заповнення працівниками бригади форми 110/о, особа якій була надана допомога дає згоду на лікування або госпіталізацію, підтверджуючи це своїм підписом. Вразі відмови від госпіталізації або транспортування пацієнт також надає свій підпис, та підписи свідків.

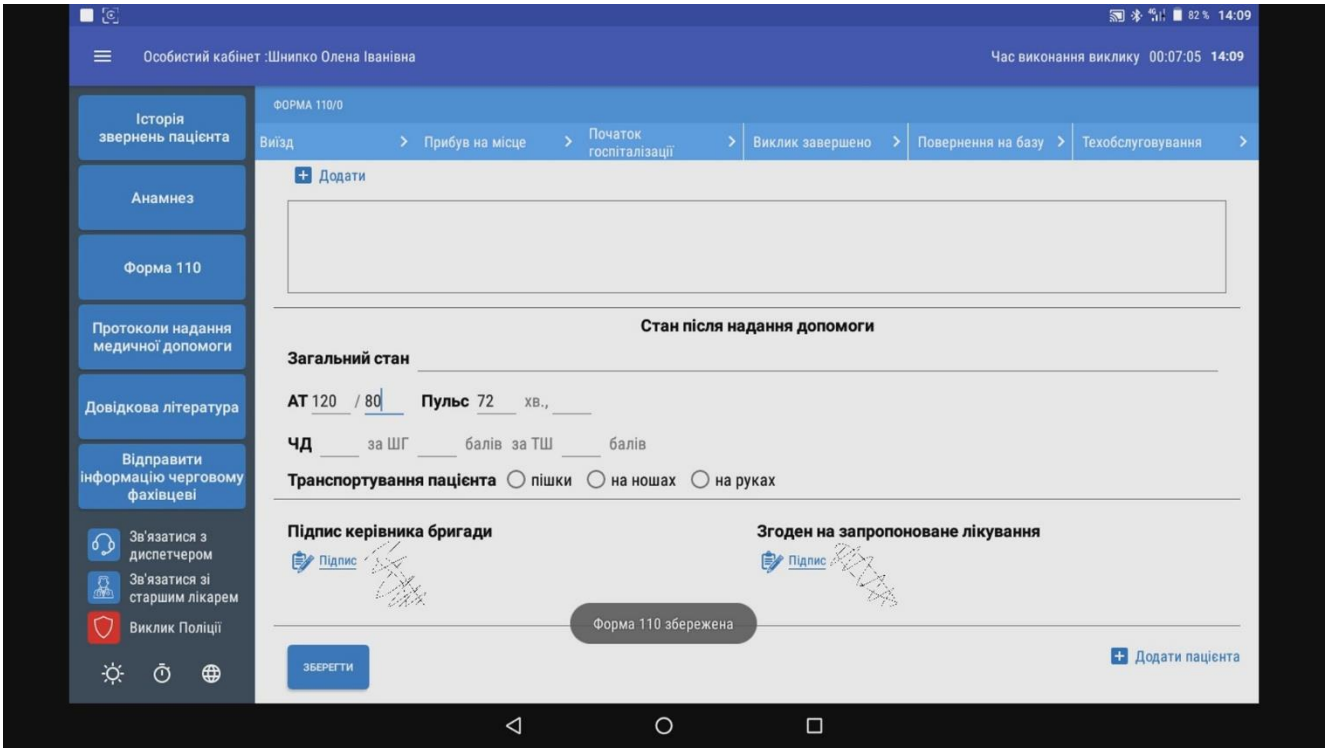


Рисунок 3.24 — Збереження та відправка форми 110/о

По завершенню заповнення форми 110/о, працівники натискають кнопку збереження «Зберегти», данні форми зберігаються в кеші додатку, а також відправляються на сервер. В разі того якщо інтернет мережа недоступна, android додаток буде відправляти дані до API автоматично з інтервалом в одну хвилину, поки не отримає код підтвердження прийому даних (статус 200).

Коли бригада покидає місце виклику, працівники мобільної бригади натискають кнопку «Виклик завершено» (рис. 3.25).

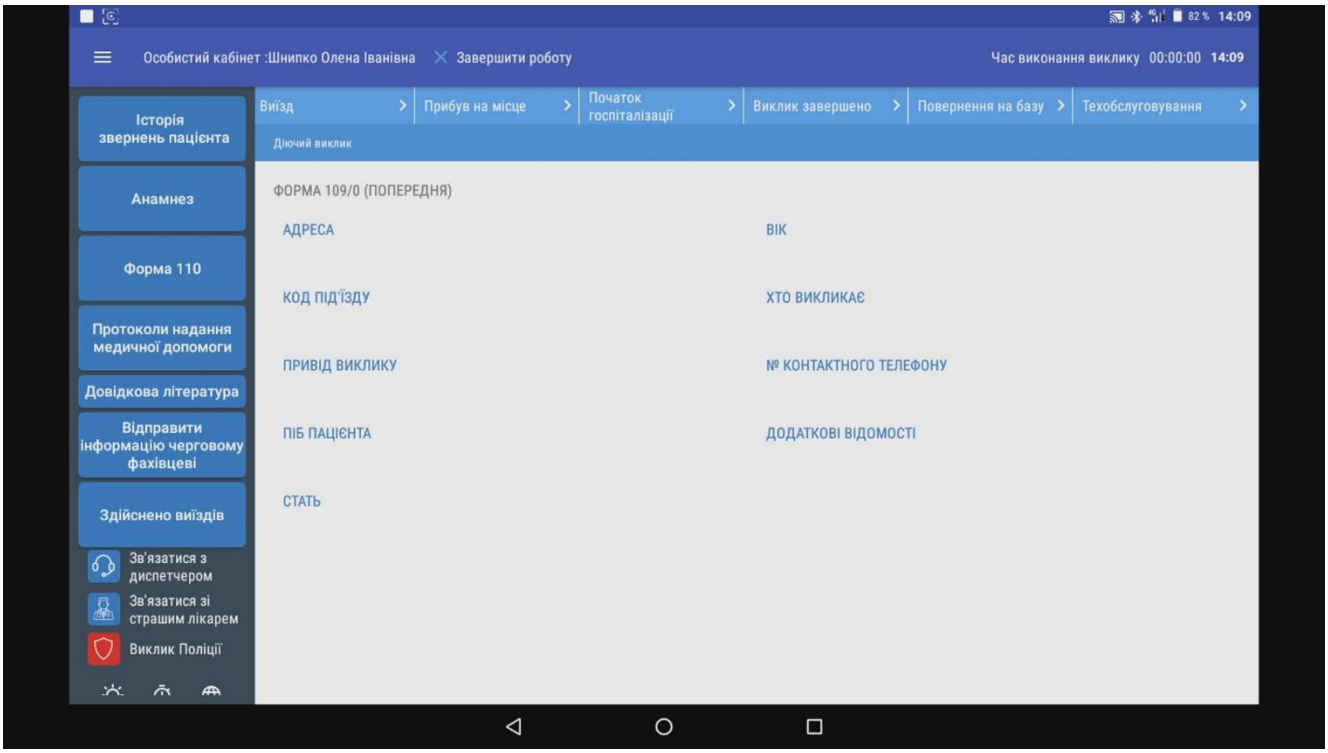


Рисунок 3.25 — Завершення виклику та очікування нового

Дані відправляються до диспетчера напрямку, та статус бригади в кабінеті «Диспетчера напрямку» (рис. 3.20) змінюється на «Вільний». Мобільна бригада знову готова до прийняття нового виклику.

Для очищення пам’яті планшета, в Android додатку «Brigada» є кнопка очищення кешу, вона дає змогу стерти с пам’яті всі підписи пацієнтів та підписи працівників мобільної бригади, які також дублюються в пам’ять пристрою та мають розширення Joint Photographic Experts Group (JPEG) та кеш заповнених форм 109/о і 110/о.

Для виклику даної функції в потрібно натиснути на клавішу «Меню», та на натиснути кнопку «Очистити Кеш» (рис. 3.26).

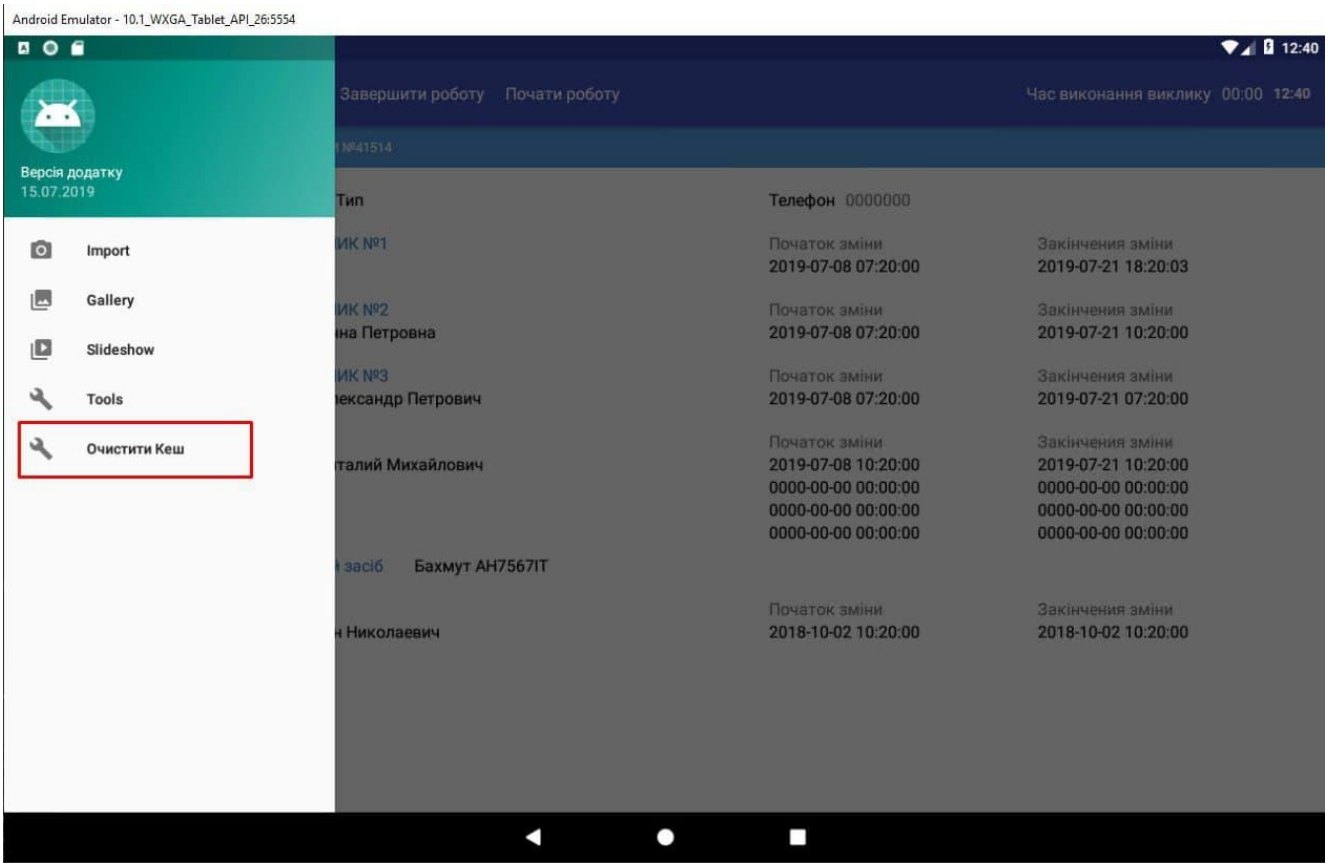


Рисунок 3.26 — Кнопка очищення кешу в Android додатку

3.3 Інтеграція Asterisk з API

Щоб система працювала та отримувала вхідні дзвінки від абонентів в серверній Швидкої медичної допомоги знаходиться сервер IP-телефонії «Asterisk».

IP-АТС Asterisk - це платформа з відкритим вихідним кодом для створення IP-телефонії «VoIP». АТС Asterisk перетворює звичайний комп'ютер або сервер в комунікаційний сервер. Asterisk підтримує системи IP PBX, шлюзи VoIP, сервери конференцій та інші спеціалізовані рішення.

Він використовується малим та великим бізнесом, центрами обробки викликів, операторами зв'язку та державними установами по всьому світу. VoIP Asterisk безкоштовний і має відкритий вихідний код. Asterisk спонсорується «Sangoma» – компанією виробником обладнання та програмного забезпечення для IP-телефонії.

Сьогодні використовується понад мільйон систем зв'язку на основі Asterisk в більш ніж 170 країнах. Asterisk використовується майже всім списком клієнтів зі списку Fortune 1000. Найчастіше він розгортається системними інтеграторами і розробниками, може стати основою для повної бізнес-телефонної системи, використовується для поліпшення або розширення існуючої системи VoIP або для подолання розриву та створення нових комунікацій між системами [6].

Також для даної IP-АТС існує велика кількість платформ та WEB-інтерфейсів від сторонніх виробників, прикладом таких являється «Elastix». Elastix - це платформа для уніфікованих комунікацій з відкритим вихідним кодом. Вона об’єднує в одному інтерфейсі IP-АТС, електронну пошту, білінг-систему, Jabber-сервер, факс-сервер, CRM-систему. В систему Elastix також входять різні засоби для організації групової роботи. Elastix має WEB-інтерфейс і включає в себе можливості, lkz організації колл-центру, аудіозапис розмов, голосову пошту, IVR, управління аудіо-конференцій (рис 3.27).

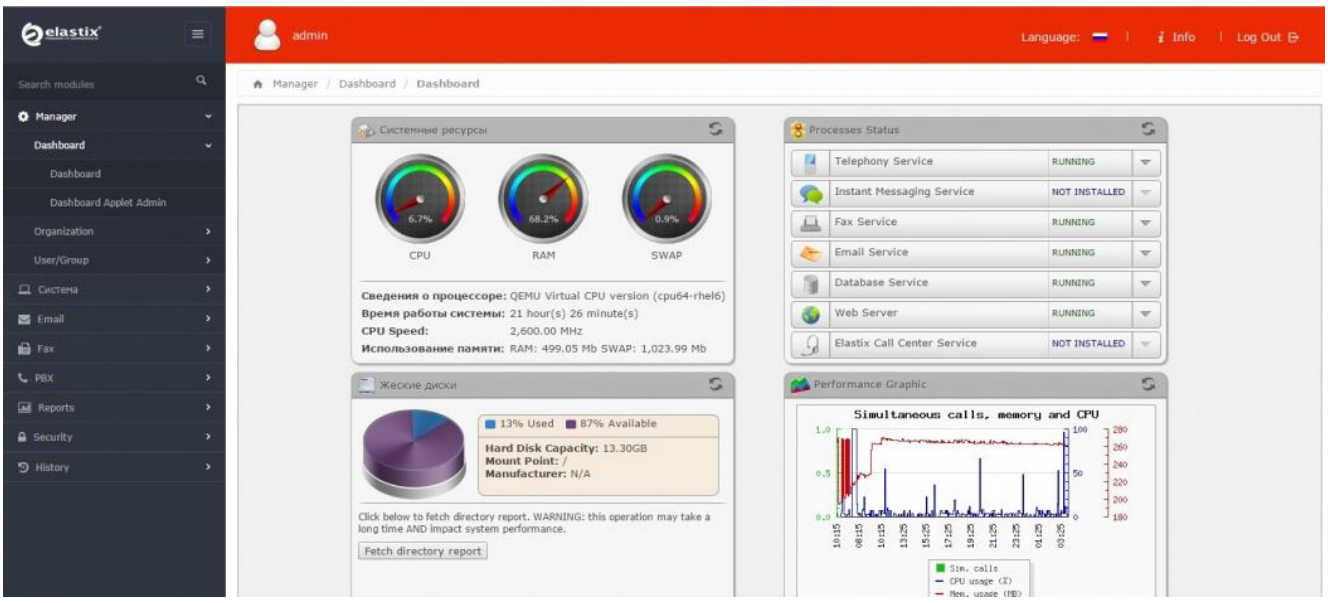


Рисунок 3.27 — Elastix інтерфейс для IP-АТС Asterisk

При здійсненні дзвінка до чергової частини Asterisk сервер створює журнал «Log» з даними. Із журналу було можливо отримувати такі дані:

- 'RINGING' — Факт вхідного дзвінка;

- 'ANSWER' — Відповідь на дзвінок;
- 'INUSE' — Телефон зайнятий, йде розмова з іншим абонентом;
- 'NOT_INUSE' — Телефон не зайнятий, від готовий отримувати вхідні дзвінки.

Було створено файл підключення та отримання інформації від Asterisk сервера під назвою «Asterisk-SWC» (рис. 3.28), більш детально код підключення буде описаний в додатку до дипломної роботи «Додаток В».

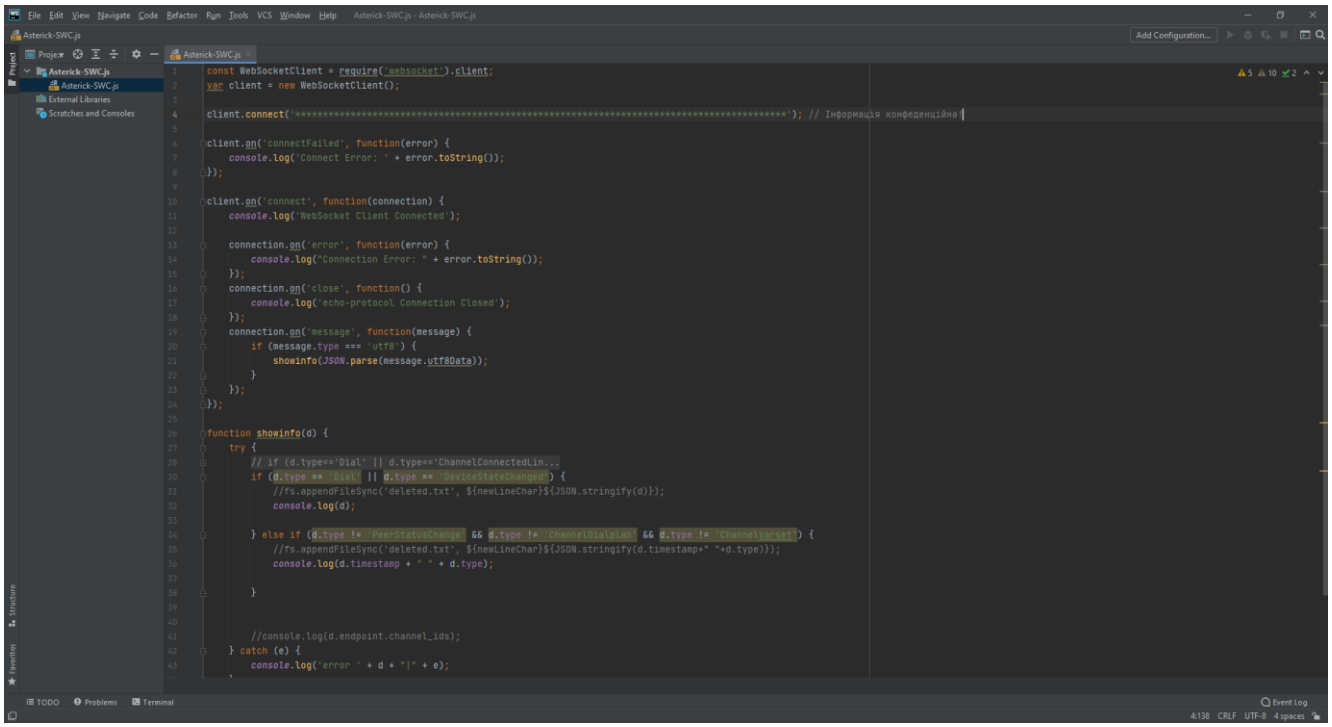


Рисунок 3.28 — Файл підключення до Asterisk сервера

Для сортування даних що надходили з сервера, був створений файл «Asterick-Object», в ньому всі дані з масиву проходили сортування, та створювалися об’єкти «PHONE» та «Phone_Call» та передаються в функцію для відправки даних.

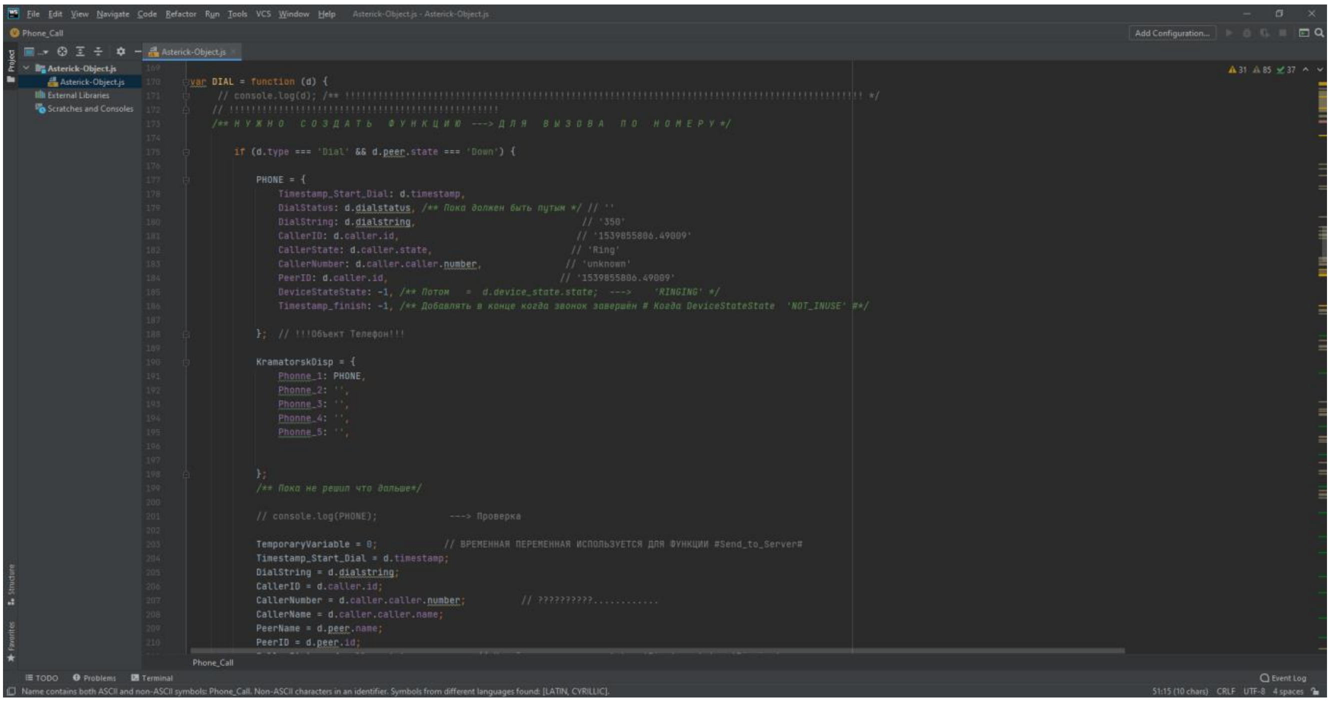


Рисунок 3.29 — Файл підключення до Asterisk сервера

Вони відповідали за дані про абонента (номер виклику, номер телефону, дата виклику, час виклику, номер SIP телефону що підняв слухавку, адреса – якщо телефонують зі стаціонарного телефону), та передавали інформацію до API (рис. 3.29).

3.4 Інструменти для розробки

JetBrains WebStorm — інтегроване середовище розробки проектів на мовах JavaScript, HTML та CSS від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA. WebStorm є спеціалізованою версією продукту PhpStorm, пропонуючи підмножину з його можливостей. WebStorm постачається з інтегрованими плаґінами JavaScript (такими як для Node.js), котрі доступні для PhpStorm безоплатно, але не входять в пакет.

WebStorm підтримує такі мови програмування: JavaScript, CoffeeScript, TypeScript та Dart. Також WebStorm забезпечує автодоповнення команд та слів-операторів, аналізуючи код програмного продукту в реальному часі, присутня

навігація по коду та по всьому проекту, рефакторинг, зневадження, присутня інтеграція з системами управління версіями.

Однією з найважливіших переваг інтегрованого середовища розробки WebStorm є робота з проектами (у тому числі, рефакторинг коду JavaScript, що міститься в різних файлах і теках проекту, а також вкладеного в HTML). Підтримується множинна вкладеність (коли в документ на HTML вкладений скрипт на Javascript, в який вкладено інший код HTML, всередині якого вкладений Javascript) — в таких конструкціях підтримується коректний рефакторинг (рис. 3.30).

Основні можливості:

- Інтеграція з системами управління версіями Subversion, Git, GitHub, Perforce, Mercurial, CVS підтримуються з коробки з можливістю побудови списку змін і відкладених змінж;
- Інтеграція з системами відстеження помилок;
- Модифікація файлів .css, html, .js з одночасним переглядом результатів (Live Edit, в деяких джерелах ця функціональність називається «редагування файлів на льоту» або «в реальному часі» або «без перезавантаження сторінки»);
- Віддалене розгортання за протоколами FTP, SFTP, на монтованих мережевих дисках тощо з можливістю автоматичної синхронізації;
- Можливості Zen Coding і Emmet.

WebStorm підтримує зневадження застосунків у node.js. Також підтримується повний набір функцій редагування застосунків на javascript — як для виконання на сервері, так і в браузері: автодоповнення, навігація по коду, рефакторінг і перевірка на помилки.

Для node.js підтримується також виведення повідомлень node.js на окрему вкладку в IDE [21].

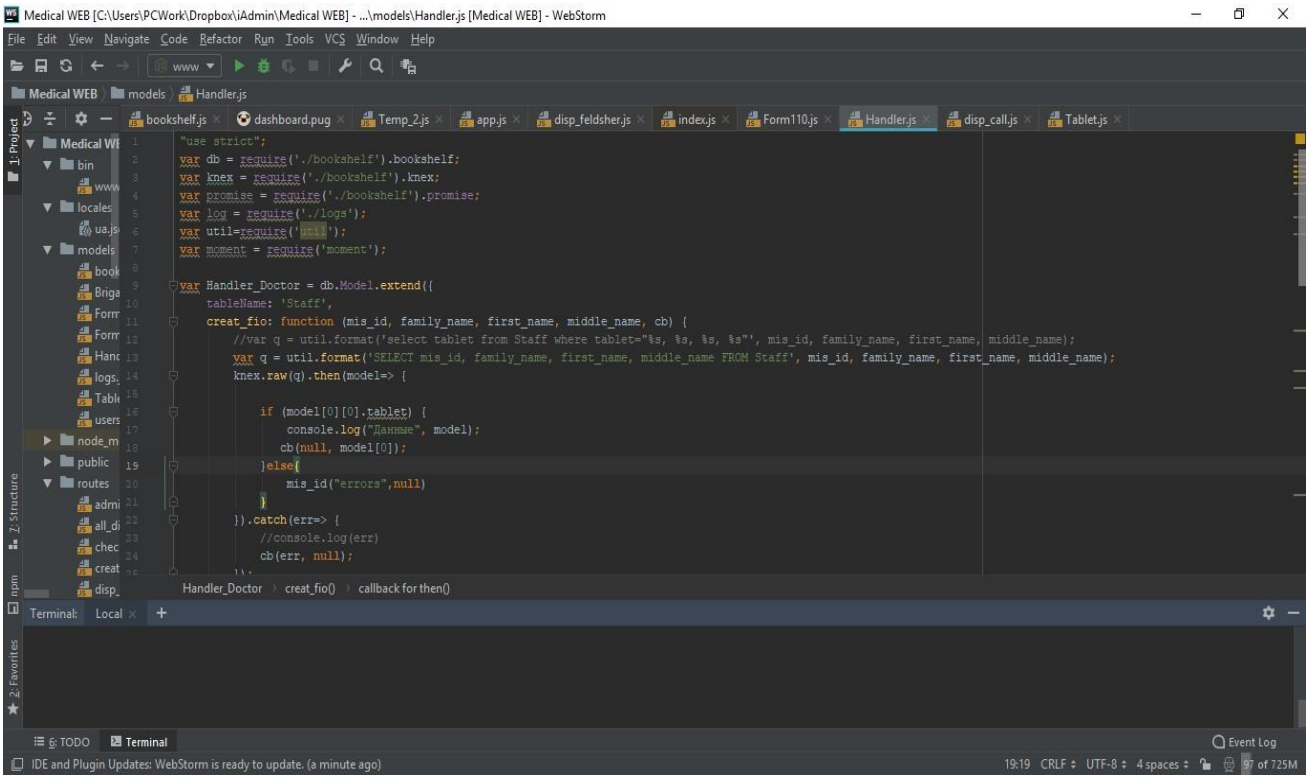


Рисунок 3.30 — Інтегроване середовище розробки JetBrains WebStorm

JetBrains PhpStorm — комерційне крос-платформове інтегроване середовище розробки для PHP, яке розробляється компанією JetBrains на основі платформи IntelliJ IDEA (рис. 3.31).

PhpStorm являє собою інтелектуальний редактор для PHP, HTML і JavaScript з можливостями аналізу коду на льоту, запобігання помилок у сирцевому коді і автоматизованими засобами рефакторинга для PHP і JavaScript. Автодоповнення коду в PhpStorm підтримує специфікацію PHP 5.3/5.4/5.5/5.6/7.0/7.1 (сучасні і традиційні проекти), включаючи генератори, співпрограми, простори імен, замикання, типажі і синтаксис коротких масивів. Присутній повноцінний SQL-редактор з можливістю редагування отриманих результатів запитів.

PhpStorm розроблений на основі платформи IntelliJ IDEA, написаної на Java. Користувачі можуть розширити функціональність середовища розробки за рахунок установки плагінів, розроблених для платформи IntelliJ, або написавши власні плагіни.

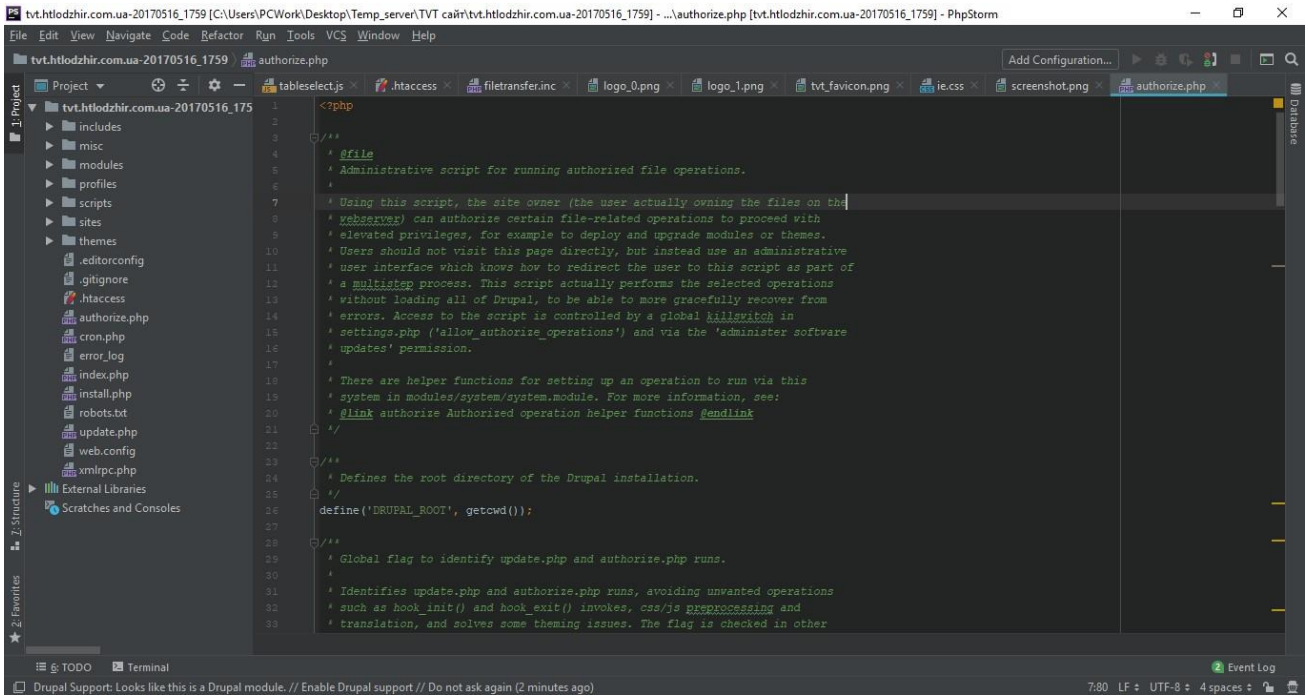


Рисунок 3.31 — Інтегроване середовище розробки JetBrains PhpStorm

Редактор коду PHP

PhpStorm надає багатий і інтелектуальний редактор коду для PHP з підсвічуванням коду, розширеною конфігурацією форматування коду, перевіркою на наявність помилок на льоту і розумним автодоповненням.

Підтримка PHP 5.3, 5.4 та 5.5, включаючи генератори, співпрограми, простори імен, замикання, типажі, синтаксис коротких масивів, доступ до члена класу при інстанціюванні, розіменування масиву при виклику функції, бінарні літерали, вираження в статичних виклики тощо. PhpStorm може використовуватися як для сучасних, так і для традиційних проектів на PHP.

Автодоповнення коду фіналізують класи, методи, імена змінних, ключові слова PHP, а також широко використовувані імена полів і змінних залежно від їхнього типу.

Підтримка стандартів оформлення коду (PSR1/PSR2, Drupal, Symfony2, Zend).

Підтримка PHPDoc. PhpStorm надає відповідне автодоповнення коду, засноване на анотаціях @property, @method і @var.

Детектор дубльованого коду.

PHP Code Sniffer (phpcs), котрий перевіряє код на льоту.

Рефакторинги (перейменування, введення змінної/константи/поля, вбудовування змінної).

Підтримка редагування шаблонів Smarty (підсвічування синтаксичних помилок, автодоповнення функцій і атрибутів Smarty, автоматична вставка парних дужок, лапок і закриваючих тегів тощо).

MVC подання для фреймворків Symfony2 і Yii.

Розпізнавання коду, запакованого в PHAR-архіви.

Середовище розробки

Підтримка SQL і баз даних (Рефакторинг схеми бази даних, генерація скриптів міграції схеми, експорт результатів виконання запиту у файл або буфер обміну, редагування збережених процедур і багато іншого).

Віддалене розгортання додатків і автоматична синхронізація з використанням FTP, SFTP, FTPS та інших протоколів.

Інтеграція з системами управління версіями (Git – включаючи спеціальний функціонал для роботи з GitHub , Subversion , Mercurial , Perforce , CVS , TFS), що дозволяє робити багато дій, наприклад commit, merge, diff та інші, прямо з PhpStorm.

Локальна історія (Local History) (локально відстежує будь-які зміни в коді). PHP UML (Діаграми класів UML для PHP коду з рефакторингом, що викликаються прямо з діаграми).

Підтримка Phing (надає автодоповнення, перевірку стандартних тегів, властивостей, імен цілей, значень атрибутів шляху в компонувальних файлах (build files).

Інтеграція з системами відстеження помилок. Підтримка Vagrant, SSH консолі і віддалених інструментів. Підтримка Google App Engine For PHP. PhpStorm також дозволяє різні поєднання клавіш для підвищення ефективності [22].

Adobe Photoshop — графічний редактор, розроблений і поширюваний фірмою Adobe Systems. Цей продукт є лідером ринку в області комерційних засобів редагування растрових зображень, і найвідомішим продуктом фірми Adobe. Часто цю програму називають просто Photoshop (Фотошоп) (рис. 3.32).

У наш час Photoshop доступний на платформах Mac OS X/Mac OS і Microsoft Windows. Ранні версії редактора були портовані під SGI IRIX, але офіційна підтримка була припинена, починаючи з третьої версії продукту. Для версії CS і CS6 можливий запуск під Linux за допомогою альтернативи Windows API — Wine.

Photoshop головним чином призначений для редагування цифрових фотографій та створення растрової графіки.

Особливості Adobe Photoshop полягають у багатому інструментарії для операції створення і обробки зображень, високій якості обробки графічних зображень, зручності й простоті в експлуатації, широких можливостях до автоматизації обробки растрових зображень, які базуються на використанні сценаріїв, механізмах роботи з кольоровими профілями, які допускають їх втілення в файли зображень з метою автоматичної корекції кольорових параметрів при виводі на друк для різних пристроїв, великому наборі команд фільтрації, за допомогою яких можна створювати найрізноманітніші художні ефекти.

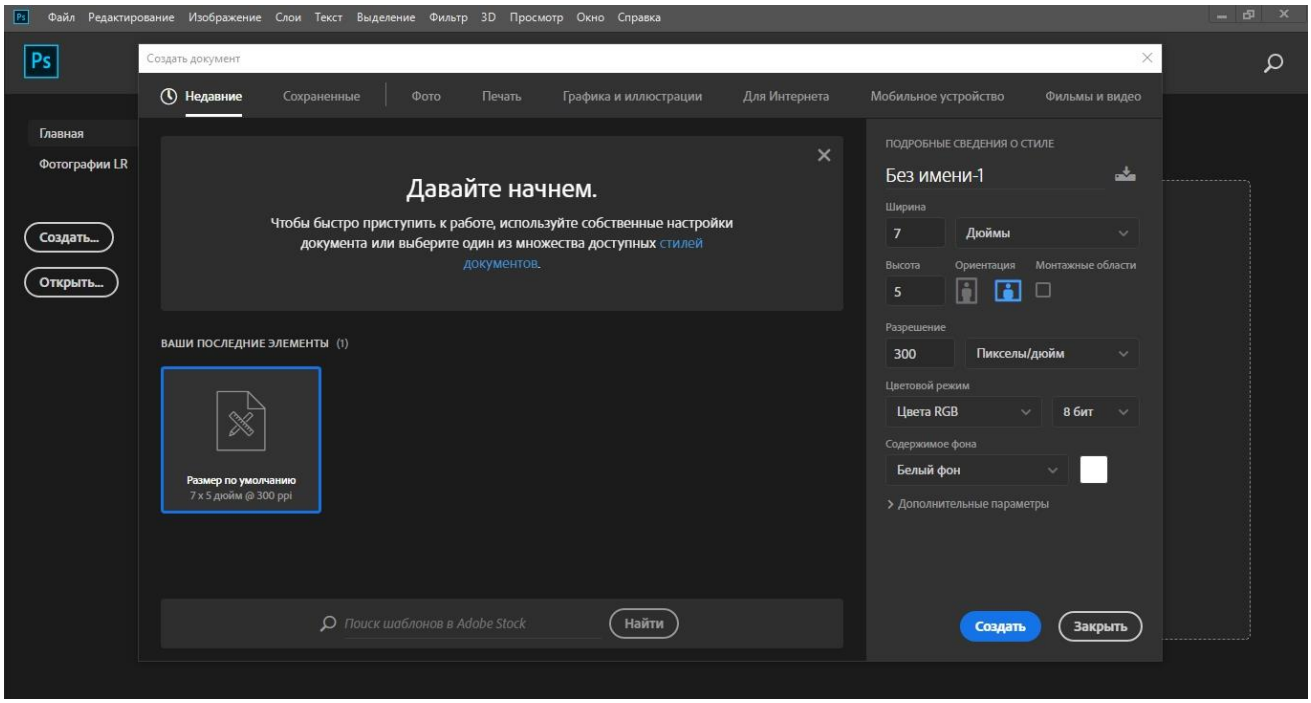


Рисунок 3.32 — Графічний редактор Adobe Photoshop CC, головна сторінка

Базові інструменти редагування дозволяють змінювати тон, насиченість зображення, обтинати його, накладати фотофільтри, виправляти перспективу тощо. Photoshop підтримує так звані шари — прозорі області зображення, на яких розміщуються елементи фотомонтажу, текст, геометричні фігури. Програма містить інструменти для роботи з текстом і нескладними фігурами, дозволяє малювати робочі контури, задавати текстам і фігурам стилі оформлення. Для роботи з окремими фрагментами зображення передбачені різні типи виділення: за фігурою, в режимі «малювання» зони виділення, за діапазоном кольорів тощо.

Photoshop головним чином призначений для редагування цифрових фотографій та створення растрової графіки.

Особливості Adobe Photoshop полягають у багатому інструментарії для операції створення і обробки зображень, високій якості обробки графічних зображень, зручності й простоті в експлуатації, широких можливостях до автоматизації обробки растрових зображень, які базуються на використанні сценаріїв, механізмах роботи з кольоровими профілями, які допускають їх втілення в файли зображень з метою автоматичної корекції кольорових

параметрів при виводі на друк для різних пристроїв, великому наборі команд фільтрації, за допомогою яких можна створювати найрізноманітніші художні ефекти (рис. 3.33).

Photoshop тісно пов'язаний з іншими програмами для обробки медіафайлів, анімації та іншої творчості. Спільно з такими програмами, як Adobe ImageReady (програма скасована у версії CS3), Adobe Illustrator, Adobe Premiere, Adobe After Effects і Adobe Encore DVD, він може використовуватися для створення професійних DVD, забезпечує засоби нелінійного монтажу і створення таких спецефектів, як фони, текстури і т. Д. для телебачення, кінематографу і всесвітньої павутини.

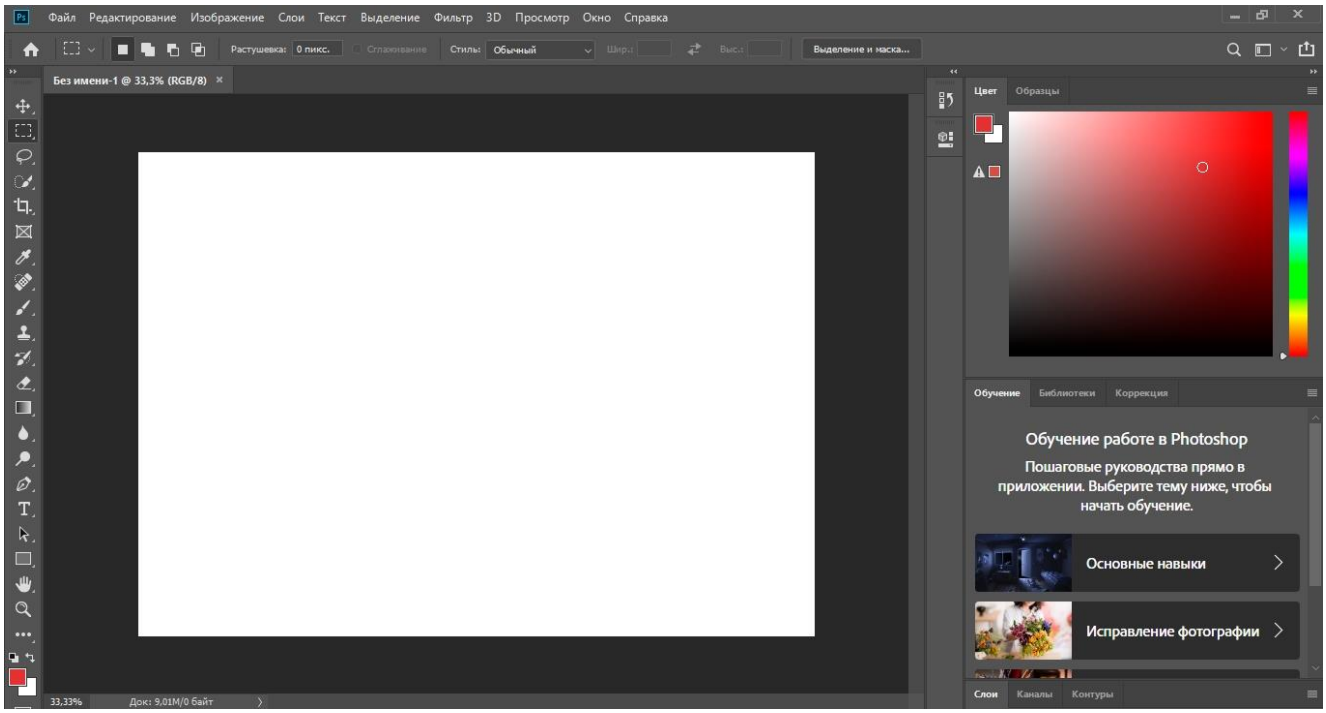


Рисунок 3.33 — Графічний редактор Adobe Photoshop CC, редактор зображення

Основний формат Photoshop, PSD, може бути експортований і імпортований всіма програмними продуктами, переліченими вище. Photoshop CS підтримує створення меню для DVD. Спільно з Adobe Encore DVD, Photoshop дозволяє створювати меню або кнопки DVD. Photoshop CS3 у версії Extended підтримує також роботу з тривимірними шарами [4].

Google Android Studio — інтегроване середовище для розробки програмного забезпечення (IDE) для платформи Android. Вперше було представлене 16 травня 2013 року на конференції Google I/O менеджером по продукції корпорації Google — Еллі Паверс. 8 грудня 2014 року компанія Google випустила перший стабільний реліз Android Studio 1.0.

Android Studio прийшла на зміну плаґіну ADT для платформи Eclipse. Середовище побудоване на базі вихідного коду продукту IntelliJ IDEA Community Edition, що розвивається компанією JetBrains. Google Android Studio розвивається в рамках відкритої моделі розробки та поширюється під ліцензією Apache 2.0.

До іншим відмінних рис даної IDE відносяться:

- Наявність безлічі помічників та шаблонів для загальних елементів програмування для Android.
- Наявність нових інструментів для упаковки і маркування коду.
- Можливість перегляду зовнішнього вигляду програми одночасно на різних пристроях Android з різними настройками і дозволом екрану.
- Висока гнучкість процесу розробки за рахунок переходу до системи автоматичного складання Gradle.

В Android Studio використовується SDK (Software development kit) — це комплект для розробки програмного забезпечення, який допомагає розробникам створювати додатки для конкретних платформ. Android SDK дає можливість розробнику легко створювати високопродуктивні додатки для смартфонів, планшетів, TV-приставок, інших спеціалізованих пристроїв (рис. 3.34).

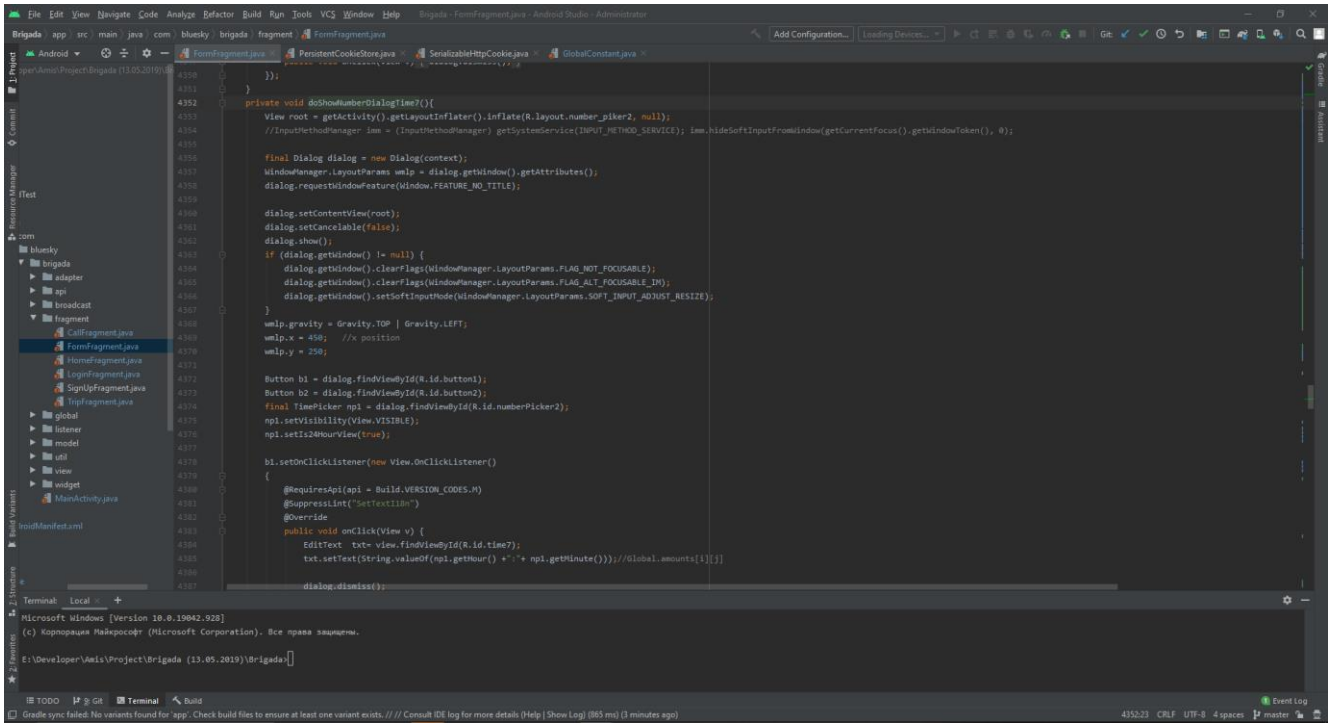


Рисунок 3.34 — Інтегроване середовище розробки Google Android Studio

Інтеграція мобільного SDK в додаток може бути надзвичайно корисною, вкрай важливо вибрати SDK який відповідає версії android, під який пишеться додаток. Два основних аспекти, властиві хорошому мобільному SDK - правильне використання даних і безпека.

Розробникам важливо, щоб SDK був не тільки високої якості, але і захищав інформацію кінцевих користувачів. Дуже важливо шукати мобільний SDK, для якого важливо дозвіл користувача про використання персональних даних, хоча зазвичай використовуються кілька бібліотек, інтегрованих з додатком.

Високоякісний мобільний SDK, який забезпечує безпеку користувачам додатку, допомагає поліпшити враження від роботи з додатком, забезпечує його надійність та безпеку.

Групи SDK:

Перші створюються розробниками для тих, хто хоче зробити самостійну програму або гри. Приклад такого SDK - DirectX, який встановлений практично на будь-якому комп'ютері. Але у простих смертних стоять тільки робочі бібліотеки - так званий Redistributable. Для програмістів ж Microsoft Corporation

підготувала повноцінний пакет DirectX SDK. У ньому є все, що необхідно розробнику для створення комп'ютерної гри: власні бібліотеки, заголовні файли для MSVC ++, приклади і багатосторінкова документація. Причому SDK поширюється абсолютно безкоштовно.

Друга група SDK - самодостатні інструменти. До них, наприклад, відноситься Torque Game Engine SDK від компанії Garage Games - повноцінний ігровий движок, що підтримує найпередовіші технології. На відміну від DirectX, який все ж напівфабрикат, з допомогою Torque можна створювати гарні ігри без глибокого знання технологій програмування під 3D.

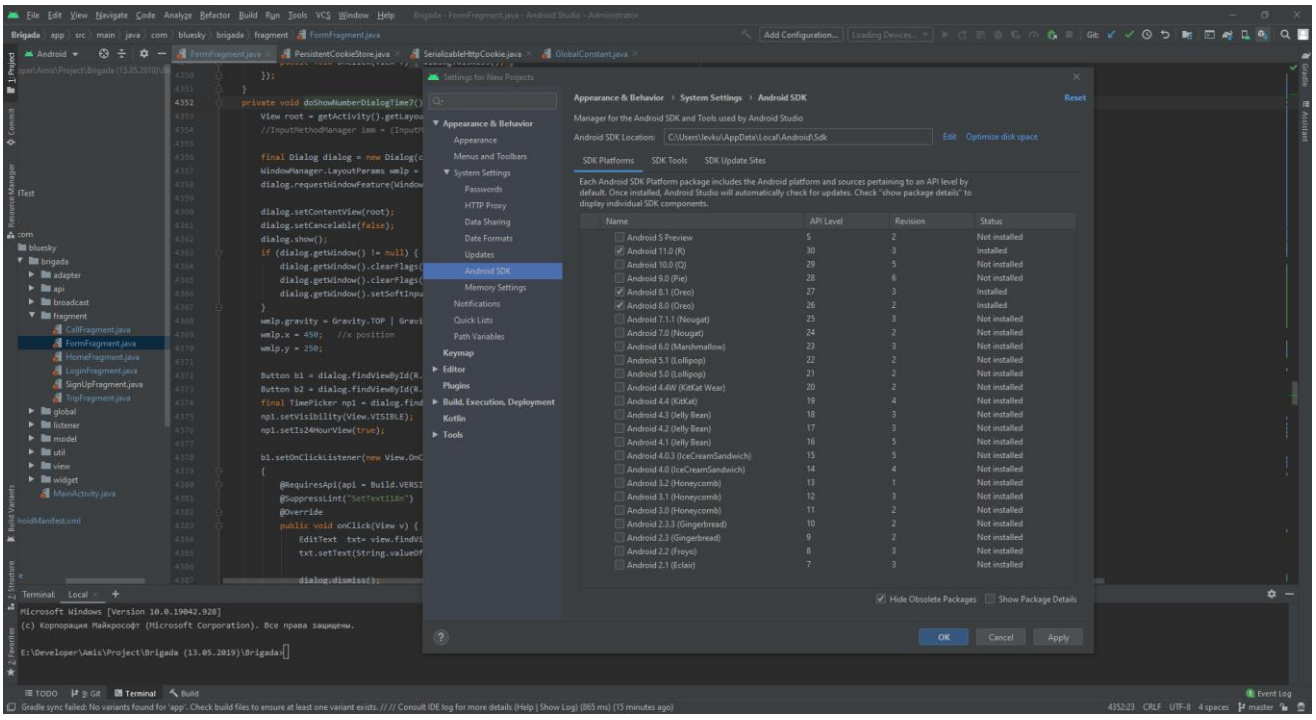


Рисунок 3.35 — Інтегрований завантажувач SDK в Android Studio

В Android Studio інтегрований завантажувач Android SDK. З його допомогою можна відразу завантажити потрібний software development kit під потрібну версію Android (рис. 3.35) [23].

3.5 Оцінка якості

Для оцінки якості API та Android додатку користувачі користуються переважно не якимись певними критеріями оцінки (тому що, швидше за все, про них взагалі не мають жодного уявлення), а абстрактними поняттями типу «хороший (або поганий)», «гарний (або жахливий)», «комфортний (або незручний)» і т. п.

Цілком природно, що називати такий підхід терміном «методика аналізу якості» не має сенсу. Тим не менш, з деякою часткою. Можна спробувати здогадатися, по яким характеристикам оцінки web-інтерфейсу API могли б бути віднесені ці висловлювання: скажімо, до його інформативності (якості контенту що відображається в формі).

Під час тестування інформаційної системи в місті Краматорськ та Маріуполь, користувачі «Диспетчери прийому» та «Диспетчери напрямку» оцінювали роботу системи, та доповідали про стабільність її роботи. Також мали змогу вносити в технічне завдання деякі побажання що відносилися до web-інтерфейсу.

Навіть тільки починаючи роботу над проектом, на стадії планування загальної структури системи працівники швидкої приймали учать у ухваленні web-інтерфейсу. Зауважу, що часто цей принцип називають ще й «загальним фактором доцільності» («загальним», тому що дія його поширюється на всі без винятку критерії оцінки API).

Розглядаючи якій би то не було проект, ми зобов'язані об'єктивно охарактеризувати його виконання безпосередньо з позиції доцільності (релевантності та якості інформації, ілюстративного наповнення, загального оформлення і так далі). Починати оцінку слід саме з мінімуму вимог.

При цьому перевіряється робота додатку з дійсними і помилковими даними, досліджується реакція програми на несподівані ситуації. Кроки процесу тестування задаються запусками програми. Запуск програми (один або кілька) повинен забезпечувати виявлення помилок, демонстрацію відповідності функцій

програми її призначенням, демонстрацію реалізації вимог характеристикам програми, відображення надійності, як індикатора якості програми

Android додаток тестувався працівниками мобільних бригад на реальних викликах в місті Краматорськ, після чого в роботу додатку були внесені деякі зміни. Річ в тім що в містах та селах Донецької області не завжди є мобільне покриття, і програма на планшет не завжди працювала коректно.

Було вирішено зробити додаток більш автономним, та дозволити йому введену інформацію зберігати в пам’яті планшету. Та відправляти коли мобільний інтернет зв’язок знову відновиться. Через те що данні постійно зберігалися на планшеті, навіть після відправлення даних на сервер, пам’ять пристрою постійно заповнювалася.

З побажать тестувальників було прийнято рішення в android додатку створити кнопку для очищення кешу, це дозволило працівникам мобільної бригади оперативно видаляти застарілу інформацію та прискорювати роботу самого планшету. Це дало змогу зробити роботу інформаційної системи стабільною.

ВИСНОВКИ

За час написання дипломної роботи було розглянуто досить важливі аспекти розробки інформаційної системи для підвищення ефективності роботи Швидкої медичної допомоги та були виконані усі поставлені задачі згідно плану магістерської дипломної роботи.

У ході теоретичного аналізу була представлена загальна структура системи яка включає в себе прикладний програмний інтерфейс (API) з web-інтерфейсам, який поділений на декілька основних ролей:

1. Роль «Диспетчера прийому» (медпрацівник, який приймає дзвінки), від громадян.
2. Роль «Диспетчера напрямку» (медпрацівник, який дивлячись на карту, де яка машина знаходиться, та її статус, направляє на машину швидкої медичної допомоги дані (Форма 109/о) — ці всі данні приходять на планшет “Android додаток”.

Проведено ряд порівнянь найпопулярніших інструментів для створення даної системи. Поміж яких було вибрано інтегроване середовище розробки JetBrains WebStorm для написання самого API на програмній платформі Node.js, інтегроване середовище розробки JetBrains PhpStorm для написання коду інтеграції API та Asterisk сервера, web-платформу Elastix для керування та адміністрування Asterisk сервера, графічний редактор Adobe Photoshop CC для створення графічних компонентів для API і Android додатку, інтегроване середовище розробки Google Android Studio для створення самого Android додатку з використанням офіційних
SDK компонентів, Navicat Premium для управління та створення бази даних та написання запитів до них.

Створена система має всі можливості, необхідні для ефективного та швидкого управління працівниками швидкої. Вона містить всі основні інструменти, які можуть знадобитися для повноцінного функціонування системи, створення звітів за потрібний період, реєстрацію нових працівників

(користувачів), налаштування індивідуального графіку роботи, управління прийнятими дзвінками, адміністрування та контроль транспортних засобів.

Система дозволить значно пришвидшити роботу закладів швидкої медичної допомоги взявши на себе майже всю роботу по комунікації між працівниками та документообігу.

На даний момент система працює в місті Краматорськ, Маріуполь, Слов’янськ.

ПЕРЕЛІК ПОСИЛАНЬ

1. Глоба Л.С. «Розробка інформаційних ресурсів та систем» 2013р. ISBN 978-617-646-173-6.

2. Репин, В. А. Бізнес процеси. Моделювання, впровадження, управління В. А. Репин -Львів: Флінта, 2013. -480 с.

3. Цуканова, О.А. Методологія та інструментарій моделювання бізнес-процесів [Текст]/ О.А. Цуканова - СПб.: Университет ИТМО, 2015, 100 с.

4. Грибов К. П. «Macromedia Flash 4.Для дизайнерів» . [Текст]/ Грибов К. П - Вид.: "Діалог-Міфі" - 1999г-295 с.

5. «Современный учебник JavaScript» [Електронный ресурс]:[Веб-сайт]– Режим доступа: https://learn.javascript.ru/ (дата звернення: 15.02.2021). – Назва з екрана

6. «Asterisk будущее телефонии Второе издание» [Текст]/ Джим Ван Меггелен, Лейф Мадсен и Джаред Сміт - Санкт-Петербург – Москва. Вид.: Символ-Плюс, 2009

7. Міністерство охорони здоров'я України «Інструкція щодо заповнення форми первинної медичної облікової документації № 109/о "Картка виклику швидкої медичної допомоги"» 2011р. [Електронний ресурс] - https://zakon.rada.gov.ua/laws/show/za148-11

8. Міністерство охорони здоров'я України «Інструкція щодо заповнення форми первинної медичної облікової документації № 110/о "Карта виїзду швидкої медичної допомоги"» 2011р. [Електронний ресурс] - https://zakon.rada.gov.ua/laws/show/za149-11

9. Міністерство охорони здоров'я України «Наказ» №999 (147/18885) 2010р. [Електронний ресурс] – https://zakon.rada.gov.ua/laws/show/z0147-11#Text

10. Барсов Р. «Побудуйте професійний сайт самі» 2009, Вид.: "Діалог-Міфі" – 2009р. -563 с.

11. Левкуша О.В. «Розробка корпоративного сайту відеонагляду на основі PHP» [Дипломна робота] – Державний університет телекомунікацій, Київ 2019р.

12. Флэнаган Д. «JavaScript. Детальне керівництво, 6-те видання.» – Пер. с англ. – СПб: Символ- Плюс, 2012. – 1080 с., іл.

13. Никсон Р. «Створюємо динамічний веб-сайт за допомогою PHP, MySQL, JavaScript, CSS и HTML5. 2 видання.» 2015р. Пітер -150-300 с.

14. Сибилёв, В.Д. Моделі і проектування баз даних: Навчальний посібник. У 2-х частинах. - Томськ: Томський міжвузівський центр дистанційної освіти – 2002. – Ч.1. 133с.

15. Печников В.Н. «Создание Web-сайтов без посторонней помощи» 2006, Пер. с англ. – СПб: Символ- Плюс, 2009. – 75 с..

16. «Php manual» [Електронний ресурс]:[Веб-сайт] – Режим доступа: https://www.php.net/manual/ru/index.php (дата звернення: 15.03.2020). – Назва з екрана.

17. Меджуи Мехди, Уайлд Эрик, Митра Ронни, Амундсен Майк «Непрерывное развитие API. Правильные решения в изменчивом технологическом» – Вид: «Питер», Санкт-Петербург, 2020р.

18. Кантелон Майк, Хартер Марк, Головайчук TJ, Райлих Натан «Node.js в действии» – Вид: «Питер», Москва, 2014р.

19. Урма Рауль-Габриэль, Фуско Марио, Майкрофт Алан «Современный язык Java. Лямбда-выражения, потоки и функциональное программирование» – Вид: «Питер», Москва, 2020р.

20. «Navicat Premium» [Електронний ресурс]:[Веб-сайт] – Режим доступа: https://www.navicat.com/ru/products/navicat-premium (дата звернення: 15.06.2020). – Назва з екрана.

21. «WebStorm» [Електронний ресурс]:[Веб-сайт] – Режим доступа: https://www.jetbrains.com/ru-ru/webstorm/features (дата звернення: 04.10.2020). – Назва з екрана.

22. «PhpStorm» Електронний ресурс]:[Веб-сайт] – Режим доступа: https://www.jetbrains.com/ru-ru/phpstorm/features (дата звернення: 05.10.2020). – Назва з екрана.

23. «Android Studio» Електронний ресурс]:[Веб-сайт] – Режим доступа: https://developer.android.com/studio/features (дата звернення: 05.10.2020). – Назва з екрана.

ДОДАТОК А. Програмний код API інтерфейсу

Оскільки програмний код API інтерфейсу дуже великий, я представлю для розгляду декілька файлів

Brigada.js

```
"use strict";
var db = require('./bookshelf').bookshelf;
var knex = require('./bookshelf').knex;
var promise = require('./bookshelf').promise;
var log = require('./logs');
var util=require('util');
var moment = require('moment');
var Brigada=db.Model.extend({
    tableName: 'brigada',
    get_brigada: function (role, text, sort_col, sort_dir, length, start, cb) {
        if (role === 2)
{//number,dispetcher_number,workplace,cause,district,st,house,corps,apartment,entrance,storey,entrance_info,phone,lastName,age,sex,whocall,dispetch_nap_number,reason_refusal,azs_departure_time,asz_return_time,call_information,time_admission,transfer_time,arrival_time,end_time,return_substation,substation_number,brigad_number,clinic_info,phone_clinic,reginter_number,accepted,handed_over,COUNT(*) as total
            var q = util.format('call select_brigada("%s","%s","%s",%d,%d)', "%" +
text + "%", sort_col, sort_dir, length, start);
            console.log(q)
            knex.raw(q).then(function (model) {
                if (model) {
                    //console.log(model[0]);
                    cb(null, model[0]);
                }
            }).catch(function (err) {
                console.log(err)
                cb(err, null);
            });
        }
    },
    create_brigada: function (role,userId,date, brigada_number,phone,type_brigada,
fio,post, timeStart, timeEnd,cleaning,driver, cars,
dateStartDriver,dateEndDriver,dateStartBrigada,dateEndBrigada, cb) {
        if (role === 9) {
            var q = util.format('call
create_brigada(%d,"%s","%s","%s","%s","%s,","%s,","%s,","%s,","%s","%s,","%s,","%s
,","%s,","%s","%s")',userId,date, brigada_number,phone,type_brigada, fio,post,
timeStart, timeEnd,cleaning,
driver,cars,dateStartDriver,dateEndDriver,dateStartBrigada,dateEndBrigada);
            console.log(q)
            knex.raw(q).then(function (model) {
                if (model) {
                    //console.log(model[0]);
                    cb(null, model[0]);
                }
            }).catch(function (err) {
                console.log(err)
                cb(err, null);
            });
        }
    },
    update_composition_brigad: function (role,userId,idd,date,
```

```
brigada_number,phone,type_brigada, fio,post, timeStart, timeEnd,cleaning,driver,
cars, dateStartDriver,dateEndDriver,dateStartBrigada,dateEndBrigada, cb) {
        if (role === 9) {
            var q = util.format('call
update_composition_brigad(%d,%d,"%s","%s","%s","%s","%s,","%s,","%s,","%s,","%s","
%s,","%s,","%s,","%s,","%s","%s")',userId,idd,date,
brigada_number,phone,type_brigada, fio,post, timeStart, timeEnd,cleaning,
driver,cars,dateStartDriver,dateEndDriver,dateStartBrigada,dateEndBrigada);
            console.log(q)
            knex.raw(q).then(function (model) {
                if (model) {
                    //console.log(model[0]);
                    cb(null, model[0]);
                }
            }).catch(function (err) {
                console.log(err)
                cb(err, null);
            });
        }
    },
    get_compositionBrigada: function (role, brigada_number, brigada_id, cb) {
        if(role === 9){
            var q = util.format('call select_composition_brigad(%d)',brigada_id);
            console.log(q)
            knex.raw(q).then(function (model) {
                if(model) cb(null, model[0])
            }).catch(function (err) {
                console.log(err)
                cb(err,null)
            })
        }
    },
    delete_peopleBrigada: function(role,userId,id,id_people,post,cb){
      if(role === 9){
          var q = util.format('call
deletePeople_brigad(%d,%d,"%s")',id,id_people,post);
          console.log(q);
          knex.raw(q).then(model =>{
              if(model) cb(null, model[0])
          }).catch(err =>{
              console.log(err)
              cb(err,null)
          })
      }
    },
    delete_driverBrigada: function(role,userId,id,id_people,cars,cb){
        if(role === 9){
            var q = util.format('call
deletedriver_brigad(%d,%d,"%s")',id,id_people,cars);
            console.log(q);
            knex.raw(q).then(model =>{
                if(model) cb(null, model[0])
            }).catch(err =>{
                console.log(err)
                cb(err,null)
            })
        }
    },
    delete_brigada: function(role,userId,id,brigada_number,cb){
        if(role === 9){
            var q = util.format('delete from brigad where id = %d and
brigada_number="%s" ',id,brigada_number);
            console.log(q);
```

```
            knex.raw(q).then(model =>{
                if(model) cb(null, model[0])
            }).catch(err =>{
                console.log(err)
                cb(err,null)
            })
        }
    },
    getBrigadNum:  function(role, serch, cb) {
        if (role === 9) {
            var q = util.format('call get_BrigadNum("%s")', serch);
            console.log(q)
            knex.raw(q).then(function (model) {
                if (model) {
                    cb(null, model[0]);
                }
            }).catch(function (err) {
                console.log(err)
                cb(err, null);
            });
        }
    },
    getBrigad_phone:function(role, serch, cb) {
        if (role === 9) {
            var q = util.format('call getBrigad_phone(%d)', serch);
            console.log(q)
            knex.raw(q).then(function (model) {
                if (model) {
                    cb(null, model[0]);
                }
            }).catch(function (err) {
                console.log(err)
                cb(err, null);
            });
        }
    },
    delete_dispetcher: function(role,userId,id,cb){
        if(role === 9){
            var q = util.format('delete from people where id = %d; ',id);
            console.log(q);
            knex.raw(q).then(model =>{
                if(model) cb(null, model[0])
            }).catch(err =>{
                console.log(err)
                cb(err,null)
            })
        }
    },
    create_dispetcher:function (role,userId, fio,post, timeStart, timeEnd,sshmd,
cb) {
        if (role === 9) {
            var q = util.format('call
create_disp(%d,"%s","%s","%s","%s","%s")',userId,fio,post, timeStart,
timeEnd,sshmd);
            console.log(q)
            knex.raw(q).then(function (model) {
                if (model) {
                    //console.log(model[0]);
                    cb(null, model[0]);
                }
            }).catch(function (err) {
                console.log(err)
                cb(err, null);
```

```
            });
        }
    },
    get_compositionDispetcher: function (role, disp, sshmd, cb) {
        if(role === 9){
            var q = util.format('call
select_composition_dispetcher("%s","%s")',disp,sshmd);
            console.log(q)
            knex.raw(q).then(function (model) {
                if(model) cb(null, model[0])
            }).catch(function (err) {
                console.log(err)
                cb(err,null)
            })
        }
    },


});

module.exports={Brigada : Brigada}
```

central103-asterisk.js

```
const WebSocketClient = require('websocket').client;
const fs = require('fs');
var clc = require('cli-color');
var moment = require('moment');
const https = require('https');
var Promise = require('bluebird');

var zvonok= function() {
  id=-1,
  start_call=-1,
  answer_call=-1,
  end_call=-1,
  status=-1,
  caller_num=-1,
  who_answer_call=-1,
  call_card_id=-1
};

var telefon = function(){
  Caller_id=-1,
  Timestamp=-1,
  Name='',
  State='NOT_INUSE'
};
var calls_count = 0;
var last_zvonok = 0;
var zvonki=[];
var telefoni=[];
var taskServer = 'sandbox.aem.org.ua:****',
    username = '***********',
    password = '******-******-******-******',
    token = '';
var client = new WebSocketClient();
client.connect('ws://***.***.***.***:****/ari/events?app=CENTRAL103&subscribeAll=t
rue&api_key=Levkusha:*******************************');
```

```
client.on('connectFailed', function(error) {
  WriteLogError('Connect Error: ' + error.toString());
});

client.on('connect', function(connection) {
  WriteLog1('WebSocket Client Connected');
      fs.readFile(__dirname +"/token", "utf8", function(err,data) {
          if(err) {
              return console.log(err);
          } else {
              token=data;
              WriteLog("Used token "+token);
          }
      });
      connection.on('error', function(error) {
        WriteLogError("Connection Error: " + error.toString());
      });

      connection.on('close', function() {
        WriteLog1('echo-protocol Connection Closed');
      });

      connection.on('message', function(message) {
            if (message.type === 'utf8') {
                showinfo(JSON.parse(message.utf8Data));
            }
      });
});

function showinfo(d) {
  try {
    //if (d.type=='Dial' || d.type=='ChannelConnectedLine' ||
d.type=='BridgeCreated' || d.type=='ChannelEnteredBridge' ||
d.type=='ChannelHangupRequest' || d.type=='BridgeDestroyed') {
      //if (d.type=='ChannelHangupRequest' || d.type=='BridgeDestroyed') {
      if (d.type=='Dial' || d.type=='DeviceStateChanged') {
        if(d.type=='DeviceStateChanged'){
          if(typeof telefoni[d.device_state.name] !== 'undefined'){

            if(d.device_state.state==="NOT_INUSE" &&
telefoni[d.device_state.name].state==="Up")  {

zvonki[telefoni[d.device_state.name].Caller_id].end_call=d.timestamp;
              zvonki[telefoni[d.device_state.name].Caller_id].status='END';
              telefoni[d.device_state.name].state=d.device_state.state;

              calls_count--;
              //console.log(zvonki);
              //console.log(telefoni);
              WriteLog('CALL END Total calls - '+calls_count+'
'+telefoni[d.device_state.name].Caller_id.toString());
              if(zvonki[telefoni[d.device_state.name].Caller_id].call_card_id!=-
1){
                  //console.log(zvonki[telefoni[d.device_state.name].Caller_id])

UpdateCallCard(zvonki[telefoni[d.device_state.name].Caller_id].call_card_id,
                                '1S1',
                                'Оператор103',

zvonki[telefoni[d.device_state.name].Caller_id].start_call,

zvonki[telefoni[d.device_state.name].Caller_id].end_call,
                                "Екстрений",
```

```
                                "Виклик бригади",
                                "Работаем в тестовом режиме");
                    }
                  } else if (d.device_state.state==="NOT_INUSE" &&
zvonki[telefoni[d.device_state.name].Caller_id].status!=='END') {
                    //telefoni[d.device_state.name].state=d.device_state.state;

                    //console.log(d);
                    //console.log(zvonki);
                    //console.log(telefoni);
                  }
                }
              }
              if(d.type=='Dial' && d.dialstatus=='CANCEL' && typeof zvonki[d.caller.id]
!== 'undefined'){
                //zvonki[d.caller.id].status=d.dialstatus;
                //calls_count--;
                WriteLog('CALL CANCEL Total calls - '+calls_count+' '+d.caller.id);
                /*if(zvonki[d.caller.id].call_card_id!=-1){
                  UpdateCallCard(zvonki[d.caller.id].call_card_id,
                                 '1S1',
                                 'Оператор103',
                                 zvonki[d.caller.id].start_call,
                                 d.timestamp,
                                 "Не визначено",
                                 "Помилковий виклик",
                                 "Работаем в тестовом режиме");
                }*/
              } else {
                //console.log(d);
              }

              if(d.type=='Dial' && d.dialstatus==''){
                if(typeof zvonki[d.caller.id] === 'undefined') {
                  // does not exist
                  if(typeof telefoni[d.peer.name.split('-')[0]] === 'undefined' ||
telefoni[d.peer.name.split('-')[0]]=='NOT_INUSE') {
                    var tf = new telefon();
                    tf.Caller_id=d.caller.id;
                    tf.name=d.peer.caller.name;
                    tf.state=d.peer.state;
                    telefoni[d.peer.name.split('-')[0]]=tf;
                    //console.log(telefoni);
                  } else {
                    telefoni[d.peer.name.split('-')[0]].Caller_id=d.caller.id;
                    telefoni[d.peer.name.split('-')[0]].name=d.peer.caller.name;
                    telefoni[d.peer.name.split('-')[0]].state=d.peer.state;
                  }
                  var zv = new zvonok();
                  zv.id=d.caller.id;
                  zv.start_call=d.timestamp;
                  zv.status=d.caller.state;
                  zv.caller_num=d.caller.caller.number;
                  zvonki[d.caller.id]=zv;
                  last_zvonok=zvonki.length;
                  delete zv;
                  delete tf;
                  //console.log(zvonki)
                  calls_count++;
                  WriteLog('NEW CALL Total calls - '+calls_count+' '+d.caller.id);
                }
                else {
                  // does exist
```

```
            zvonki[d.caller.id].status=d.caller.state;
            //console.log(zvonki);
          }
        }
        if(d.type=='Dial' && d.dialstatus=='ANSWER'){
          if(typeof zvonki[d.caller.id] === 'undefined') {
            // does not exist
          }
          else {
            // does exist
            zvonki[d.caller.id].status=d.dialstatus;
            zvonki[d.caller.id].answer_call=d.timestamp;
            zvonki[d.caller.id].who_answer_call=d.peer.name;
            WriteLog('ANSWER CALL Total calls - '+calls_count+' '+d.caller.id);
            //console.log(zvonki);
            //console.log(telefoni);
CreateCallCard('1S1',d.caller.id,zvonki[d.caller.id].caller_num,zvonki[d.caller.id
].start_call,zvonki[d.caller.id].answer_call);
            if(typeof telefoni[d.peer.name.split('-')[0]] === 'undefined' ||
telefoni[d.peer.name.split('-')[0]]=='NOT_INUSE') {
              var tf = new telefon();
              tf.Caller_id=d.caller.id;
              tf.name=d.peer.caller.name;
              tf.state=d.peer.state;
              telefoni[d.peer.name.split('-')[0]]=tf;
            } else{
              telefoni[d.peer.name.split('-')[0]].state=d.peer.state;
            }
            //console.log(zvonki);
            //console.log(telefoni);
          }
        }
        //console.log(d);
      //if(d.dialstatus == undefined) {
      //  console.log(d);
      //}
      } else if (d.type!='PeerStatusChange' && d.type!='ChannelDialplan' &&
d.type!='ChannelVarset') {
        //fs.appendFileSync('deleted.txt',
`${newLineChar}${JSON.stringify(d.timestamp+" "+d.type)}`);
        //console.log(d.timestamp+" "+d.type);
    }
  } catch (e){
    console.log('error '+d+"|"+e);
  }
}
function
CreateCallCard(operator_id,mis_call_card_id,caller_number,start_datetime,end_datet
ime) {
    var carrent_Startdatetime = start_datetime.split("+");
    var carrent_Enddatetime = end_datetime.split("+");
    var getDate = start_datetime.split("T")
    var card = mis_call_card_id.split(".")
    var MisCallCard ="CARD-"+getDate[0]+"-"+ card[0]
  SendPostCmd('/api/callcard/',{
          "CallCard": [
            {
            "operator_id": operator_id,
            "mis_call_card_id": MisCallCard,
            "caller_number": (caller_number === "unknown")? "0": caller_number ,
            "call_station": "PBX",
            "start_datetime": carrent_Startdatetime[0],
            "end_datetime": carrent_Enddatetime[0]
```

```
            }
        ]
    })
    .then(function (resp) {
        JSON.parse(resp,function (key,val) {
            //console.log(val)
            if(key=='call_card_id') {
              zvonki[mis_call_card_id].call_card_id=val;
            }
        })
        WriteLog("CallCard created "+mis_call_card_id);
    })
    .catch(function (err) {
        WriteLogError('CallCard create ошибка: '+err);
      fs.readFile(__dirname +"/token", "utf8", function(err,data) {
            if(err) {
               return console.log(err);
            } else {
               token=data;
               WriteLog("Used token "+token);
            }
          });
    })
}
function
UpdateCallCard(call_card_id,operator_id,call_station,start_datetime,end_datetime,a
ll_priority,call_result,call_comment) {
    var carrent_Startdatetime = start_datetime.split("+");
    var carrent_Enddatetime = end_datetime.split("+");
    if(call_card_id !== undefined && call_card_id !== "undefined" && call_card_id
!== null) {
        SendPutCmd('/api/callcard/' + call_card_id, {
            "CallCard": [
                {
                    "operator_id": operator_id,
                    "call_station": call_station,
                    "start_datetime": carrent_Startdatetime[0],
                    "end_datetime": carrent_Enddatetime[0],
                    "call_priority": all_priority,
                    "call_result": call_result,
                    "call_comment": call_comment,

                }
            ]
        })
            .then(function (resp) {
                JSON.parse(resp, function (key, val) {

                })
                WriteLog("CallCard updated /api/callcard/" + call_card_id);
            })
            .catch(function (err) {
                WriteLogError('CallCard update ошибка: ' + err);

            })
    }
}
function WriteLog(text) {
  var dt = moment().format('YYYY-MM-DD HH:mm:ss');
  console.log('%s | %s\r', dt,text);
}
function WriteLogError(text) {
  var dt = moment().format('YYYY-MM-DD HH:mm:ss');
```

```
  console.log(dt+clc.red(' | '+text));
}
function WriteLog1(text) {
  var dt = moment().format('YYYY-MM-DD HH:mm:ss').replace(/T/, ' ');
  console.log(dt+clc.green(' | '+text));
}
function WriteLog2(text) {
  var dt = moment().format('YYYY-MM-DD HH:mm:ss').replace(/T/, ' ');
  console.log(dt+clc.yellow(' | '+text));
}

function SendPostCmd(addr, text) {
  return new Promise(function (resolve,reject) {
      var postData = JSON.stringify(text);
      console.log("SendPostCmd : "+ addr+" "  +postData)
      var answer = undefined;

      var options1 = {
          hostname: taskServer.split(':')[0],
          port: taskServer.split(':')[1],
          path: addr,
          method: 'POST',
          rejectUnauthorized : false,
          headers: {
              'Content-Type': 'application/json',
              'Content-Length': Buffer.byteLength(postData),
              'Authorization': 'JWT '+token
          }
      };

      var req = https.request(options1, function(res){
          res.setEncoding('utf8');
          res.on('data', function(data){
                  if(data){
                      answer = data;
                      //WriteLog1('Got answer: '+data);
                  } else {

                  }
          });
          res.on('end', function(){
              if(res.statusCode==400 || res.statusCode==401)
reject(res.statusCode);
              resolve(answer);
          });

      }).on('error', function(err){
          reject(err);
      })

      req.write(postData,'utf8',function () {
          //WriteLog('Sent: '+postData);
      });
      req.end();
  })

}

function SendPutCmd(addr, text) {
  return new Promise(function (resolve,reject) {
      var postData = JSON.stringify(text);
      console.log("SendPutCmd : "+ addr+" "  +postData)
      var answer = undefined;
```

```
        var options1 = {
            hostname: taskServer.split(':')[0],
            port: taskServer.split(':')[1],
            path: addr,
            method: 'PUT',
            rejectUnauthorized : false,
            headers: {
                'Content-Type': 'application/json',
                'Content-Length': Buffer.byteLength(postData),
                'Authorization': 'JWT '+token
            }
        };

        var req = https.request(options1, function(res){
            res.setEncoding('utf8');
            res.on('data', function(data){
                    if(data){
                        answer = data;
                        //WriteLog1('Got answer: '+data);
                    } else {

                    }
            });
            res.on('end', function(){
                if(res.statusCode==400 || res.statusCode==401)
reject(res.statusCode);
                resolve(answer);
            });

        }).on('error', function(err){
            reject(err);
        })

        req.write(postData,'utf8',function () {
            //WriteLog('Sent: '+postData);
        });
        req.end();
    })

}
```

ДОДАТОК Б. Програмний код Android додатку

Оскільки програмний код API інтерфейсу дуже великий, я представлю для розгляду декілька файлів

HomeFragment.java

```
package com.bluesky.brigada.fragment;
import android.os.Bundle;
import android.support.v7.widget.DefaultItemAnimator;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import com.bluesky.brigada.MainActivity;
import com.bluesky.brigada.R;
import com.bluesky.brigada.adapter.DriverRecyclerInfoAdapter;
import com.bluesky.brigada.adapter.DriverRecyclerViewAdapter;
import com.bluesky.brigada.util.Global;
import com.bluesky.brigada.util.Utils;
public class HomeFragment extends android.support.v4.app.Fragment  implements
View.OnClickListener {
    private View view;
    public HomeFragment() {}
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View root = inflater.inflate(R.layout.fragment_home, container, false);
        this.view = root;
        initFragment();
        return view;
    }
    @Override
    public void onClick(View v) {
        int id = v.getId();
        switch (id) {
            case R.id.button_warehouse:
                Utils.mainFragmentLoad(new HomeFragment(), R.id.fragment_content);
                break;
            case R.id.button_exit_trip:
                Utils.mainFragmentLoad(new TripFragment(), R.id.fragment_content);
                break;
        }
    }
    private void initFragment() {
        ((TextView)
Global.parent.findViewById(R.id.text_head_title)).setText("Особистий кабинет : "+
((MainActivity)this.getActivity()).getResp_user().fio);
        ((TextView) view.findViewById(R.id.txt_home_header)).setText("СКЛАД
БРИГАДИ №"+ ((MainActivity)this.getActivity()).getResp_user().brigad_number);
        ((TextView) view.findViewById(R.id.txt_home_brigad_number)).setText(
((MainActivity)this.getActivity()).getResp_user().brigad_number);
        ((TextView) view.findViewById(R.id.txt_home_type)).setText(
((MainActivity)this.getActivity()).getResp_user().type);
        ((TextView) view.findViewById(R.id.txt_home_phone)).setText(
((MainActivity)this.getActivity()).getResp_user().phone);
```

```
        ((TextView) view.findViewById(R.id.txt_home_med1_fio)).setText(
((MainActivity)this.getActivity()).getResp_user().med1.fio);
        ((TextView) view.findViewById(R.id.txt_home_med1_start_time)).setText(
((MainActivity)this.getActivity()).getResp_user().med1.dateTime_start);
        ((TextView) view.findViewById(R.id.txt_home_med1_end_time)).setText(
((MainActivity)this.getActivity()).getResp_user().med1.dateTime_end);
        ((TextView) view.findViewById(R.id.txt_home_med2_fio)).setText(
((MainActivity)this.getActivity()).getResp_user().med2.fio);
        ((TextView) view.findViewById(R.id.txt_home_med2_start_time)).setText(
((MainActivity)this.getActivity()).getResp_user().med2.dateTime_start);
        ((TextView) view.findViewById(R.id.txt_home_med2_end_time)).setText(
((MainActivity)this.getActivity()).getResp_user().med2.dateTime_end);
        ((TextView) view.findViewById(R.id.txt_home_med3_fio)).setText(
((MainActivity)this.getActivity()).getResp_user().med3.fio);
        ((TextView) view.findViewById(R.id.txt_home_med3_start_time)).setText(
((MainActivity)this.getActivity()).getResp_user().med3.dateTime_start);
        ((TextView) view.findViewById(R.id.txt_home_med3_end_time)).setText(
((MainActivity)this.getActivity()).getResp_user().med3.dateTime_end);
            RecyclerView driverFioRecyclerView =
view.findViewById(R.id.driver_fio_recyclerview);
            RecyclerView driverInfoRecyclerView =
view.findViewById(R.id.driver_info_recyclerview);
            DriverRecyclerViewAdapter driverRecyclerViewAdapter = new
DriverRecyclerViewAdapter(this, ((MainActivity)
this.getActivity()).getResp_user().driver, R.layout.recyclerview_row_driver);
            driverFioRecyclerView.setAdapter(driverRecyclerViewAdapter);
            driverFioRecyclerView.setLayoutManager(new
LinearLayoutManager(this.getContext()));
            driverFioRecyclerView.setItemAnimator(new DefaultItemAnimator());
            DriverRecyclerInfoAdapter driverInfoRecyclerViewAdapter = new
DriverRecyclerInfoAdapter(this, ((MainActivity)
this.getActivity()).getResp_user().driver, R.layout.recyclerview_row_driver_info);
            driverInfoRecyclerView.setAdapter(driverInfoRecyclerViewAdapter);
            driverInfoRecyclerView.setLayoutManager(new
LinearLayoutManager(this.getContext()));
            driverInfoRecyclerView.setItemAnimator(new DefaultItemAnimator());
        ((TextView) view.findViewById(R.id.txt_home_vehicle)).setText(
((MainActivity)this.getActivity()).getResp_user().vehicle);
        ((TextView) view.findViewById(R.id.txt_home_train_fio)).setText(
((MainActivity)this.getActivity()).getResp_user().trainee.fio);
        ((TextView) view.findViewById(R.id.txt_home_train_start_time)).setText(
((MainActivity)this.getActivity()).getResp_user().trainee.dateTime_start);
        ((TextView) view.findViewById(R.id.txt_home_train_end_time)).setText(
((MainActivity)this.getActivity()).getResp_user().trainee.dateTime_end);

Global.parent.findViewById(R.id.button_finish).setVisibility(View.VISIBLE);
        view.findViewById(R.id.button_warehouse).setOnClickListener(this);
        view.findViewById(R.id.button_exit_trip).setOnClickListener(this);
    }
}
```

LoginFragment.java

```
package com.bluesky.brigada.fragment;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.design.widget.TextInputEditText;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.LayoutInflater;
import android.view.View;
```

```
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.TextView;
import com.bluesky.brigada.MainActivity;
import com.bluesky.brigada.R;
import com.bluesky.brigada.api.ApiClient;
import com.bluesky.brigada.api.ApiInterface;
import com.bluesky.brigada.global.GlobalConstant;
import com.bluesky.brigada.model.Resp_User;
import com.bluesky.brigada.util.Global;
import com.bluesky.brigada.util.Utils;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import org.json.JSONException;
import org.json.JSONObject;
import java.lang.reflect.Type;
import java.util.HashMap;
import java.util.Map;
import okhttp3.ResponseBody;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
public class LoginFragment extends android.support.v4.app.Fragment implements
View.OnClickListener, TextWatcher {
    private View view;
    public LoginFragment() {
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View root = inflater.inflate(R.layout.fragment_login, container, false);
        this.view = root;
        initFragment(view);
        return view;
    }
    @Override
    public void onClick(View v) {
        int id = v.getId();
        switch (id) {
            case R.id.button_download:
                Utils.mainFragmentLoad(new HomeFragment(), R.id.fragment_content);
                break;
            case R.id.button_leave:
                login(this.view);
                break;
        }
    }
    private void login(View v) {
        TextInputEditText inputUserEditText = (TextInputEditText)
v.findViewById(R.id.edit_username);
        TextInputEditText inputPasswordEditText = (TextInputEditText)
v.findViewById(R.id.edit_password);
        final String userName = inputUserEditText.getText().toString();
        final String password = inputPasswordEditText.getText().toString();
        ApiInterface apiInterface =
ApiClient.getClient(this.getActivity()).create(ApiInterface.class);
        HashMap<String, Object> param = new HashMap<String, Object>();
        param.put("role", "5");
        param.put("deviceId", GlobalConstant.ANDROID_ID);
        param.put("login", userName);
        param.put("pass", password);
        apiInterface.login(GlobalConstant.CONTENT_TYPE, param).enqueue(new
```

```
Callback<ResponseBody>() {
            @Override
            public void onResponse(Call<ResponseBody> call, Response<ResponseBody>
response) {
                Gson gson=new Gson();
                int code = response.code();
                JSONObject object = null;
                if (code == 200) {
                    ResponseBody responseBody = response.body();
                    try {
                        Resp_User
resp_user=gson.fromJson(responseBody.string(),Resp_User.class);
                        String user_name = resp_user.fio;
                        if (user_name != null) {
                            Utils.showShortToast(LoginFragment.this.getActivity(),
"Login Successfully");
((MainActivity)LoginFragment.this.getActivity()).setResp_user(resp_user);
                            try {
                                SharedPreferences preferences =
LoginFragment.this.getActivity().getSharedPreferences(GlobalConstant.APP_PREFERENC
E_NAME, Context.MODE_PRIVATE);
                            SharedPreferences.Editor editor = preferences.edit();

                            editor.putBoolean(GlobalConstant.PREFERENCE_IS_LOGIN,
true);
                            editor.putString(GlobalConstant.PREFERENCE_USER_NAME,
userName);
                            editor.putString(GlobalConstant.PREFERENCE_PASSWORD,
password);
                                editor.commit();
                            }
                            catch (Exception ex) {
                                 ex.printStackTrace();
                            }
((MainActivity)LoginFragment.this.getActivity()).findViewById(R.id.button_start).s
etVisibility(View.VISIBLE);
                            //send_request_brigada_param();
                    Utils.mainFragmentLoad(new HomeFragment(),
R.id.fragment_content);
                        }
                    } catch (Exception e) {
                        try {
                            if(object==null) {
                     Utils.showShortToast(LoginFragment.this.getActivity(), "server
error")
                                return;
                            }
                            String status = object.getString("errors");
                            Utils.showShortToast(LoginFragment.this.getActivity(),
status);
                        } catch (JSONException e1) {
                            e1.printStackTrace();
                        }
                        e.printStackTrace();
                    }

                } else {
                  Utils.showShortToast(LoginFragment.this.getActivity(), "Login
faield");
                }
            }
            @Override
            public void onFailure(Call<ResponseBody> call, Throwable t) {
```

```
                Utils.showShortToast(LoginFragment.this.getActivity(), "Login
faield");
            }
        })
    }
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after)
{
    }
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
    }
    @Override
    public void afterTextChanged(Editable s) {
        String str = s.toString();
        ((TextView) view.findViewById(R.id.text_count)).setText(String.format("%d
/ 50", str.length()));
    }
    private void initFragment(View view) {

        this.view.findViewById(R.id.button_download).setOnClickListener(this);
        this.view.findViewById(R.id.button_leave).setOnClickListener(this);
        ((EditText)
this.view.findViewById(R.id.edit_username)).addTextChangedListener(this);
        try {
            SharedPreferences preferences =
this.getActivity().getSharedPreferences(GlobalConstant.APP_PREFERENCE_NAME,
Context.MODE_PRIVATE);
            boolean isLoggedIn =
preferences.getBoolean(GlobalConstant.PREFERENCE_IS_LOGIN, false);
            if (isLoggedIn) {

                String username =
preferences.getString(GlobalConstant.PREFERENCE_USER_NAME, "");
                String password =
preferences.getString(GlobalConstant.PREFERENCE_PASSWORD, "");
                TextInputEditText inputUserEditText = (TextInputEditText)
view.findViewById(R.id.edit_username);
                TextInputEditText inputPasswordEditText = (TextInputEditText)
view.findViewById(R.id.edit_password);
                inputUserEditText.setText(username);
                inputPasswordEditText.setText(password);
                //login(view);
            }
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ/НАВЧАЛЬНО НАУКОВИЙ ІНСТИТУТ ЗАОЧНОГО ТА ДИСТАНЦІЙНОГО НАВЧАННЯ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тема: РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ РОБОТИ ШВИДКОЇ МЕДИЧНОЇ ДОПОМОГИ

Виконав: студент групи ППЗМ-71
Левкуша Олександр Віталійович
Науковий керівник:
Доцент кафедри ІПЗ
Жебка Вікторія Вікторівна

Київ 2021

2

- **МЕТА РАБОТИ**: СТВОРЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ПІДВИЦЕННЯ ЕФЕКТИВНОСТІ РОБОТИ ШВИДКОЇ МЕДИЧНОЇ ДОПОМОГИ ТА АВТОМАТИЗАЦІЇ ДОКУМЕНТООБІГУ ШЛЯХОМ РОЗРОБКИ ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ ТА ANDROID ДОДАТКУ.
- **ОБ'ЄКТ ДОСЛІДЖЕННЯ**: АНАЛІЗ РОБОТИ ШВИДКОЇ МЕДИЧНОЇ ДОПОМОГИ.
- **ПРЕДМЕТ ДОСЛІДЖЕННЯ**: ПРОЦЕС ДІЙ ПРАЦІВНИКІВ ШВИДКОЇ МЕДИЧНОЇ ДОПОМОГИ ПРИ ОТРИМАННІ ВХІДНОГО ДЗВІНКА ВІД ГРОМАДЯН.

3

Постановка задачі

- Аналіз роботи швидкої медичної допомоги.
- Аналіз роботи «Диспетчерів прийому»
- Аналіз роботи «Диспетчерів напрямку»
- Аналіз роботи «Працівників мобільної бригади».
- Розробка загальної структури системи.
- Створення технічного завдання.
- Створення дизайну.
- Створення API та Android додатку.

4

Загальна структура системи

- Система включає в себе API, та Android додаток. API поділена на декілька ролей:
  - 1. Для «Диспетчера прийому» — медпрацівник, який приймає дзвінки, від громадян. (API інтегрований с системою IP телефонії Asterisk).
  - 2. Для «Диспетчера напрямку» — медпрацівник, який дивлячись на карту, де яка машина знаходиться, направляє їй дані (Форми 109/о), звідки їм надійшов виклик «Прізвище, ім'я, вік, код МКХ-10, адресу, контактні номера, номер під'їзду, пароль від домофону, та приблизний опис проблеми, яку диспетчери прийому отримали с телефонної розмови.») — ці всі данні приходять на планшет ''Android додаток''.
  - 3. Для «Головного Фельдшера» — особа яка назначає зміни працівникам медичної допомоги, тим самим даючи їм змогу авторизуватися в системі. (Якщо медпрацівник не знаходиться на зміні, він не має доступу до системи).
  - 4. «Голова відділу» — медпрацівник, який має змогу прослуховувати дзвінки громадян до ШМД, дивитись статистику викликів, «Які надійшли до ШВД, кількість прийнятих, пропущених (перенаправленних до інших відділів ШМД), кількість надання консультацій, кількість виїздів та в які саме райони, тощо».
  - 5. «Роль для медпрацівників які споряджають машини ШМД лікарськими засобами» — бригада ШМД коли приїжджає на виклик до хворого, заповнює (Форму 110), в якій вказуються всі симптоми та стан пацієнта, а також що вони використали для покращення його стану, або реанімації (перчатки, таблетки, маски, шприци, ампули, тощо…), це дозволяє працівникам розуміти що потрібно додати в комплектацію, та підраховувати витрати.
- В Android додатку реалізована робота медпрацівників мобільної бригади:
  - Отримавши дані на планшет мобільна група виїжджає на виклик, та заповнює всю необхідну інформацію на планшеті (Форма 110/о).

5

Вигляд форми 109/о


Ідентифікаційний код за ЄДРПОУ

Міністерство охорони здоров'я України
Найменування закладу

Медична облікова документація
Форма N 109/о
Наказ МОЗ
17.11.2010 N 999

ЧАС
Прийому: годин ___ хвилин ___
Передачі: годин ___ хвилин ___
Виїзду: годин ___ хвилин ___
Прибуття: годин ___ хвилин ___
Закінчення: годин ___ хвилин ___
Повернення на підстанцію: годин ___ хвилин ___

"___" ___________ 20__ року N ___________
Диспетчер N______ Робоче місце N _____
Привід до виклику: ___________________

ПЕРЕДАНО
підстанції N ___________
бригаді ШМД N ___________
у поліклініку, пункт невідкладної медичної допомоги, інше (вказати) ___________
тел. ___________
реєстр. N ___________
Прийняв: ___________
Передав: ___________

Адреса виклику:
Район (масив) ___________ вул. ___________
буд. ______ корп. _____ кв. ____ під'їзд ____ поверх ____ Вхід з: вулиці двору
тел. ___________ Прізвище ___________
Вік ________ Стать Ч/Ж Хто викликає ___________

Диспетчер напрямку N________

Додаткові відомості, причина відмови: ___________

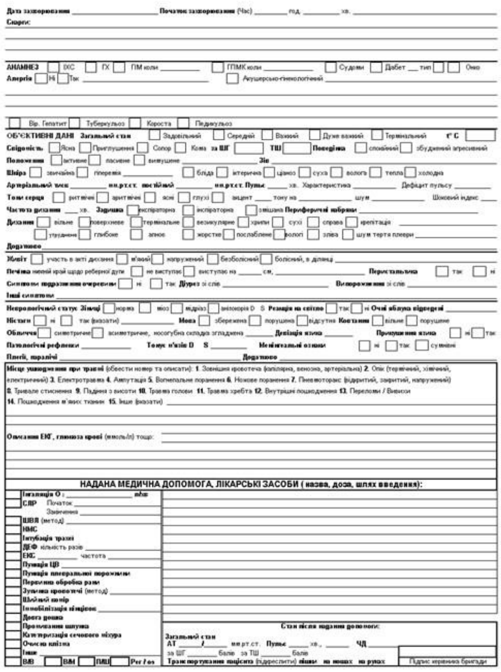
АЗС: Час виїзду; Час повернення

Довідки про стан виконання виклику: Час; Диспетчер N____

6

Вигляд форми 110/о

Сторона №1



Сторона 2

7

Вибір засобів розробки та проектування

✓HTML
✓CSS
✓JavaScript
✓jQuery
✓Pug

JetBrains WebStorm

✓PHP

JetBrains PhpStorm

✓MySQL

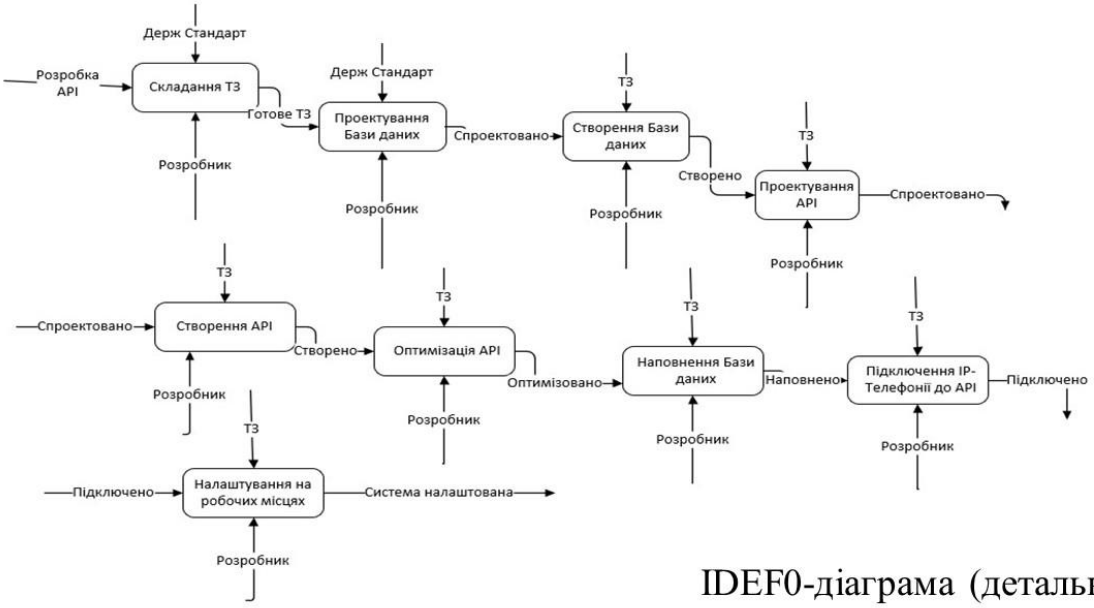
Navicat Premium

✓Java
✓Xml

Android Studio

8

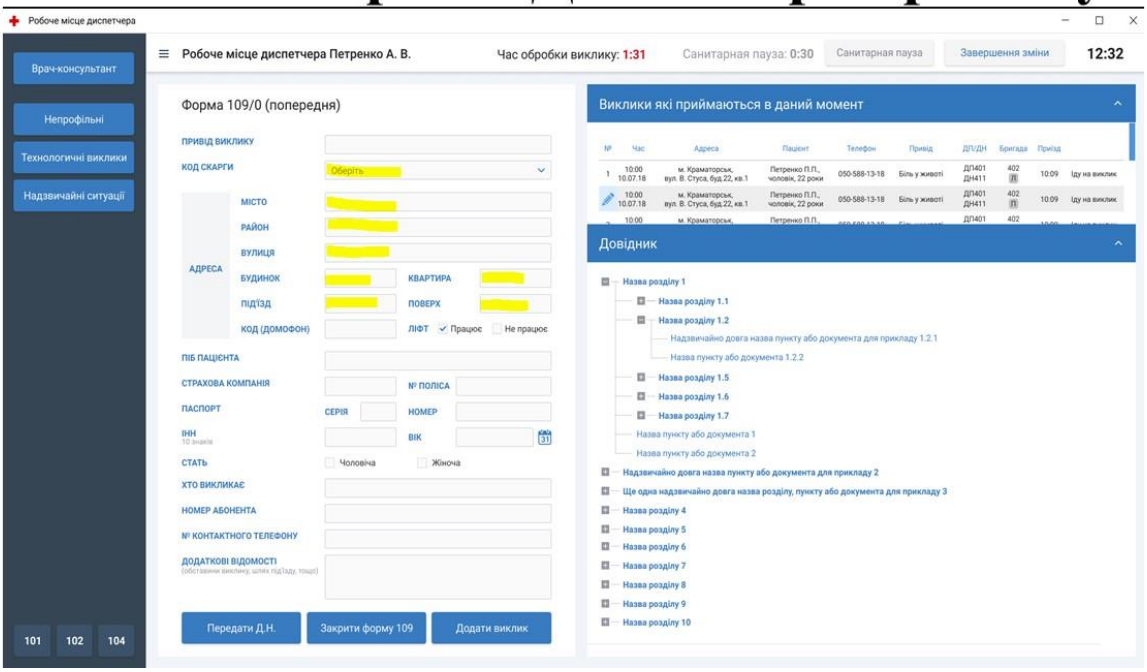
Проектування API



IDEF0-діаграма (детальна)

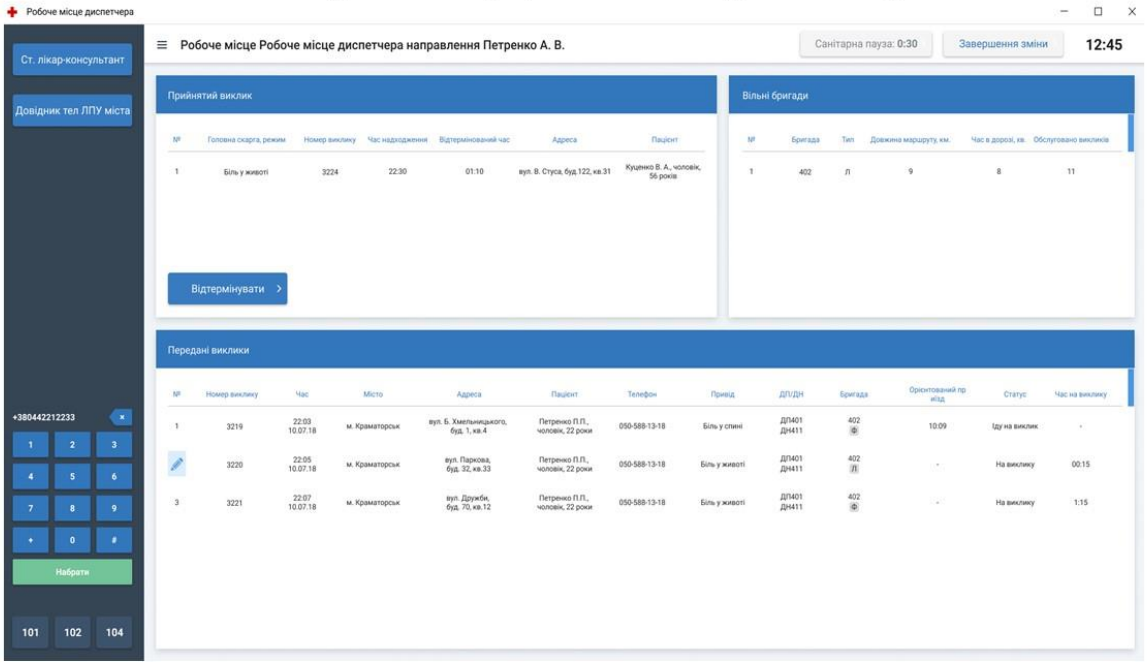
9

Головна сторінка Диспетчера прийому

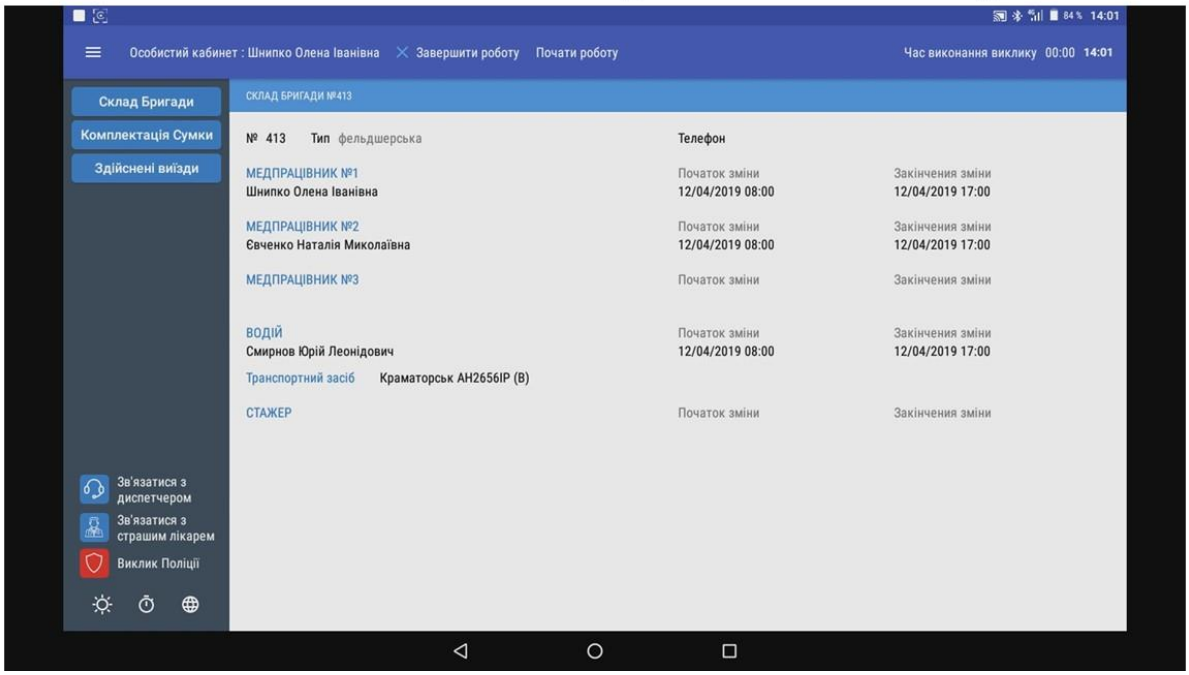


10

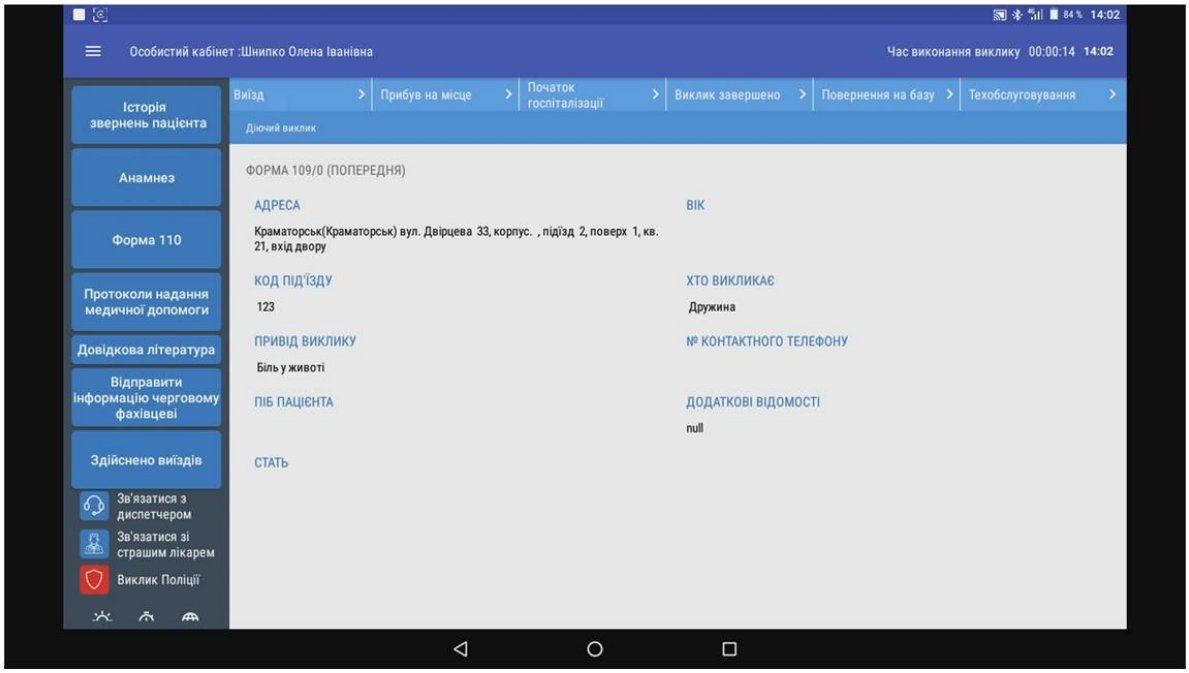
Головна сторінка Диспетчера направлення



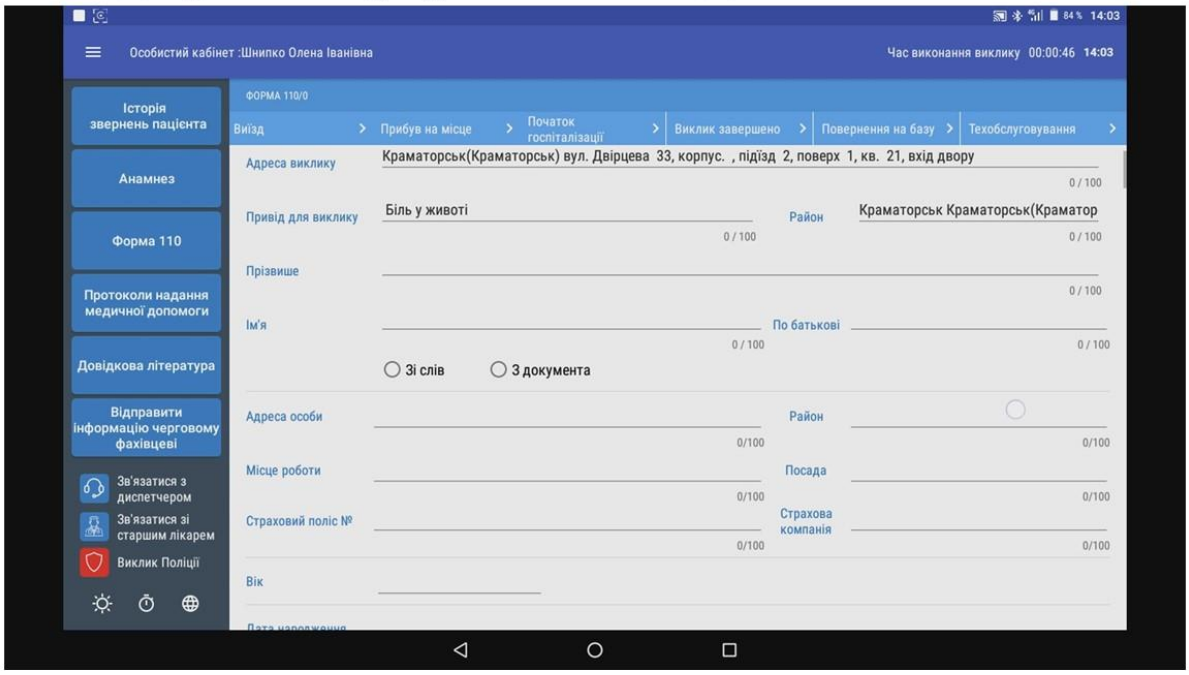
11 # Особистий кабінет лікаря мобільної бригади



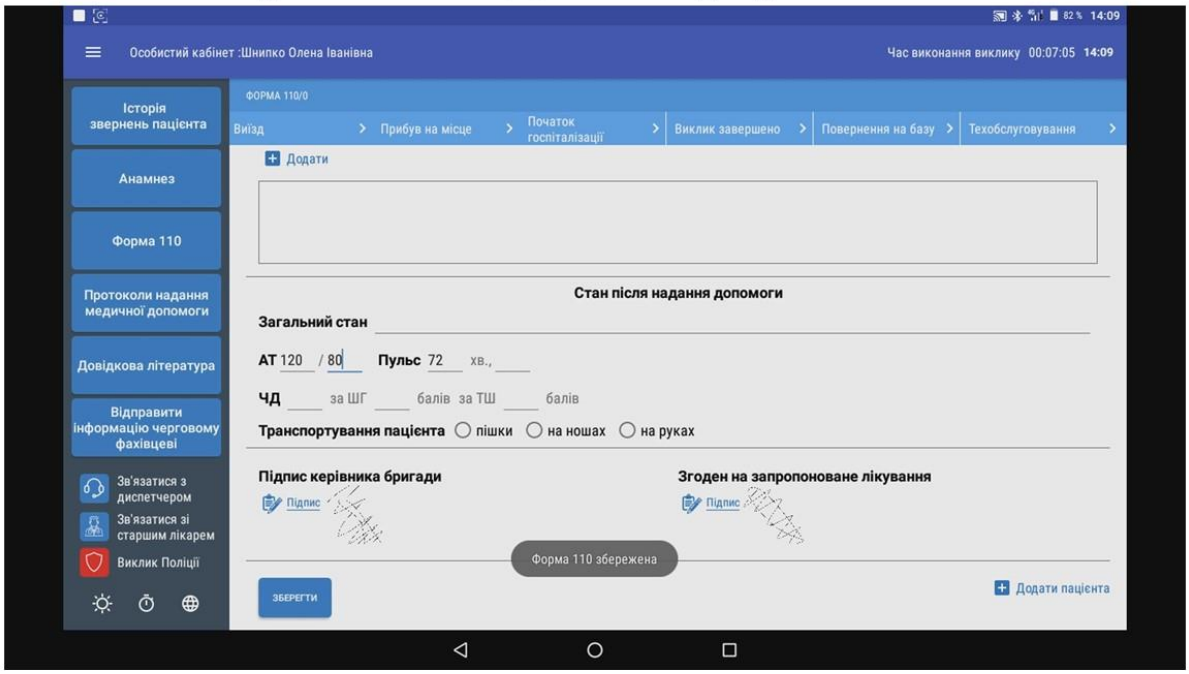
12 # Отримання даних на Android додаток



13 # Відкриття форми 110/о для заповнення

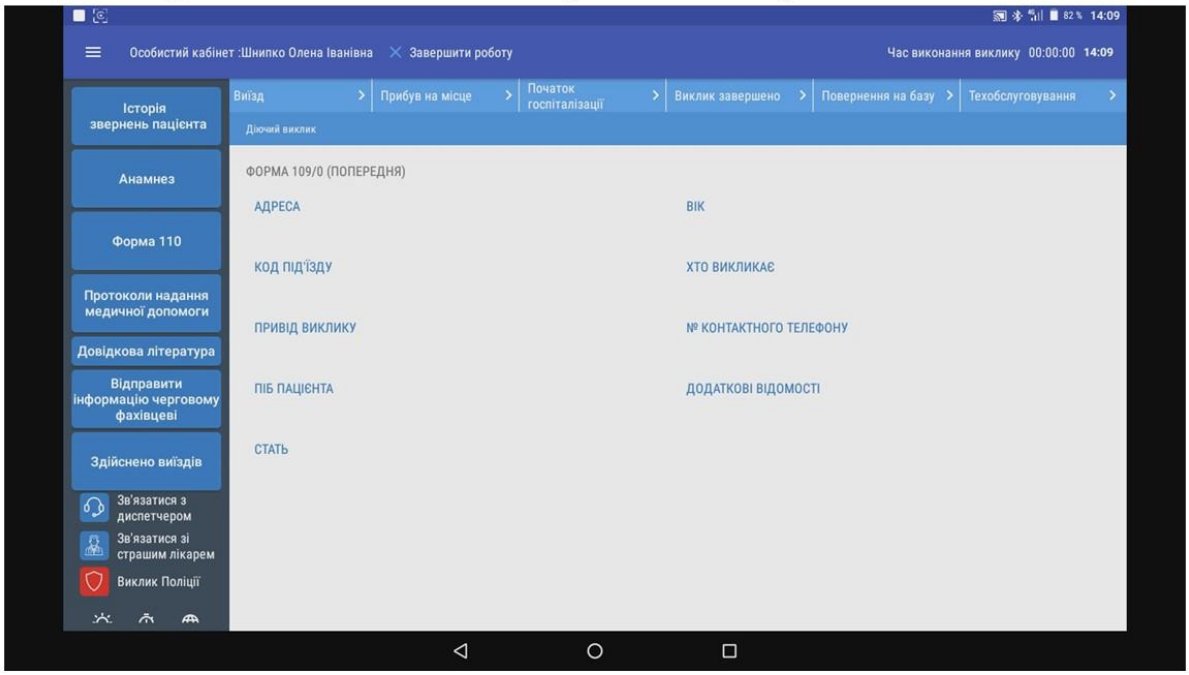


14 # Збереження даних форми 110/о



15

Завершення виклику та очікування нового



16

Висновки

- За час написання дипломної роботи було розглянуто досить важливі аспекти розробки інформаційної системи для підвищення ефективності роботи Швидкої медичної допомоги та були виконані усі поставлені задачі згідно плану магістерської дипломної роботи.
- У ході теоретичного аналізу була представлена загальна структура системи яка включає в себе прикладний програмний інтерфейс (API) з web-інтерфейсам, та Android додаток. Були описані такі ролі:
  - 1. Роль «Диспетчера прийому»;
  - 2. Роль «Диспетчера напрямку»;
  - 3. Роль «Працівника мобільної бригади».
- Проведено ряд порівнянь найпопулярніших інструментів для створення даної системи.
- Створена система має всі можливості, необхідні для ефективного та швидкого управління працівниками швидкої.

17

Висновки

- Вона містить всі основні інструменти, які можуть знадобитися для повноцінного функціонування системи, створення звітів за потрібний період, реєстрацію нових працівників (користувачів), налаштування індивідуального графіку роботи, управління прийнятими дзвінками, адміністрування та контроль транспортних засобів.
- Система дозволить значно пришвидшити роботу закладів швидкої медичної допомоги взявши на себе майже всю роботу по комунікації між працівниками та документообігу.

На даний момент система проходить тестування в містах Краматорськ, Маріуполь, Слов'янськ.

Дякую за увагу!