Міністерство освіти та науки України
Харківський національний університет радіоелектроніки
Кафедра програмної інженерії

Лабораторна робота № 4

З дисципліни: «Архітектура програмного забезпечення»

на тему: «Програмна система для автоматизації процесів прибирання приміщень»

Виконав
ст. гр. ПЗПІ-18-2
Кузнецов Роман Олександрович

Перевірив
ст. викл. каф. ПІ
Сокорчук Ігор Петрович

Харків 2021

Мета: розробити мобільну частину для програмної системи для автоматизації процесів прибирання приміщень.

Хід роботи:

В якості мобільної платформи обрано ОС Android. Для розробки мобільної частини застосунку використано мову програмування Java та середовище розробки Android Studio. Для взаємодії з серверною частиною системи використовується HTTPS протокол та JSON формат транспортування даних, що у HTTP термінології позначається як application/json. Для реалізації цієї взаємодії використовується бібліотека retrofit 2.0.

Перед тим, як програмно реалізувати мобільну частину, було проаналізовано предметну область та встановлено всі основні способи використання клієнтського застосунку. Створено UseCase діаграму, що описує сценарій поведінки застосунку у процесі взаємодії з його користувачами. UseCase діаграма наведена у додатку А.

Мобільна частина програмної системи має декілька видів акторів: людина, що є представником підрядних працівників надання послуг з прибирання приміщень, та людина, що є власником приміщень та бажає автоматизувати їх процеси прибирання. Підрядник має змогу додавати і маніпулювати послугами, які він буде надавати, має доступ до перегляду та моніторингу вже складених угод на прибирання з власниками приміщень. Власник приміщень має можливість шукати у його місцевості доступних йому підрядних працівників, переглядати та обирати їх послуги, складати з ними угоди на прибирання, переглянути вже складені, маніпулювати профілем.

Для відображення робочих компонентів мобільної частини системи та відображення логіки їх взаємодії та інженерних рішень під час проектування було створено діаграму компонентів мобільного застосунку програмної системи. Діаграма компонентів зображена на рисунку 1.

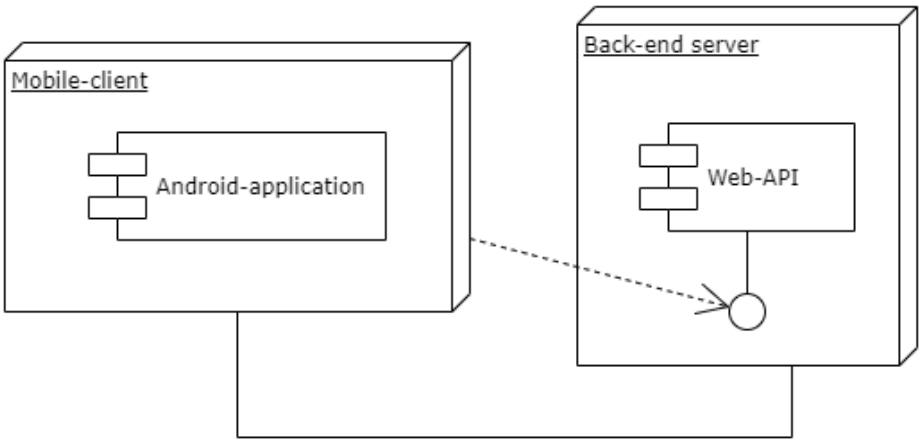

Рисунок 1 – Діаграма компонентів для мобільного застосунку програмної системи автоматизації процесів прибирання приміщень

З іншого боку було побудовано діаграму, що визначає зміну станів об'єкту у часі – діаграму станів, зображену на рисунку 2. Діаграму побудовано з точки зору системи під час використання її власником приміщень.

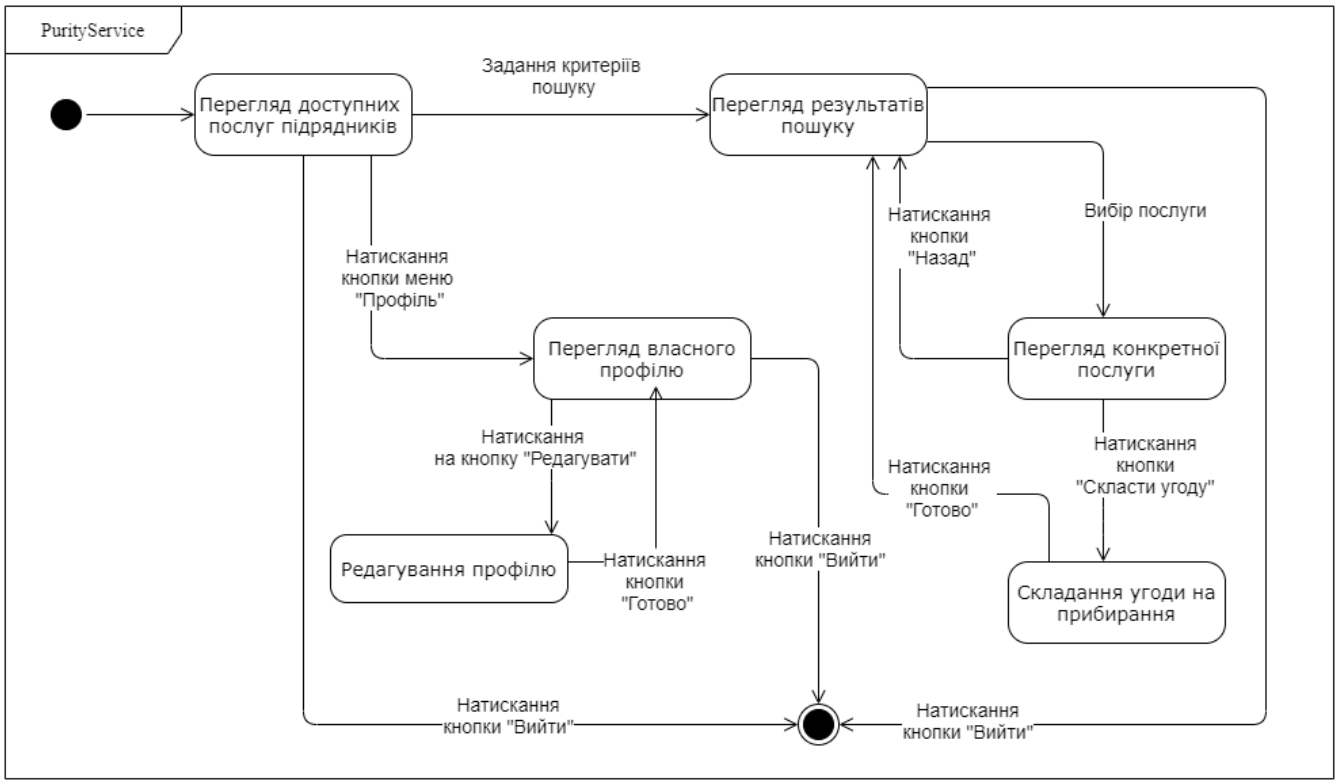

Рисунок 2 – Діаграма станів та переходів для мобільного застосунку програмної системи автоматизації процесів прибирання приміщень

Для більш детального опису умов переходів системи з одного стану в інший також побудовано діаграму діяльності, яка наведена у додатку Б.

У додатку В наведено частину програмної реалізації відправки запиту на реєстрацію користувача у системі. У додатку Г наведено частину програмної реалізації активності та запиту складання угоди на прибирання між підрядником та власником приміщень.

Посилання на архів з програмним кодом та файлом контрольної суми: drive.google.com/drive/folders/18inV6Lsb-mOXJST4a_HUMSNjxrlPRNfT?usp=sharing

Контрольна сума до архіву: 7359caba44ec8e15e0093af451c5c2df

Висновок: під час виконання лабораторної роботи було розроблено mobile частину для програмної системи автоматизації процесів прибирання приміщень.

ДОДАТОК А

UseCase діаграма системи

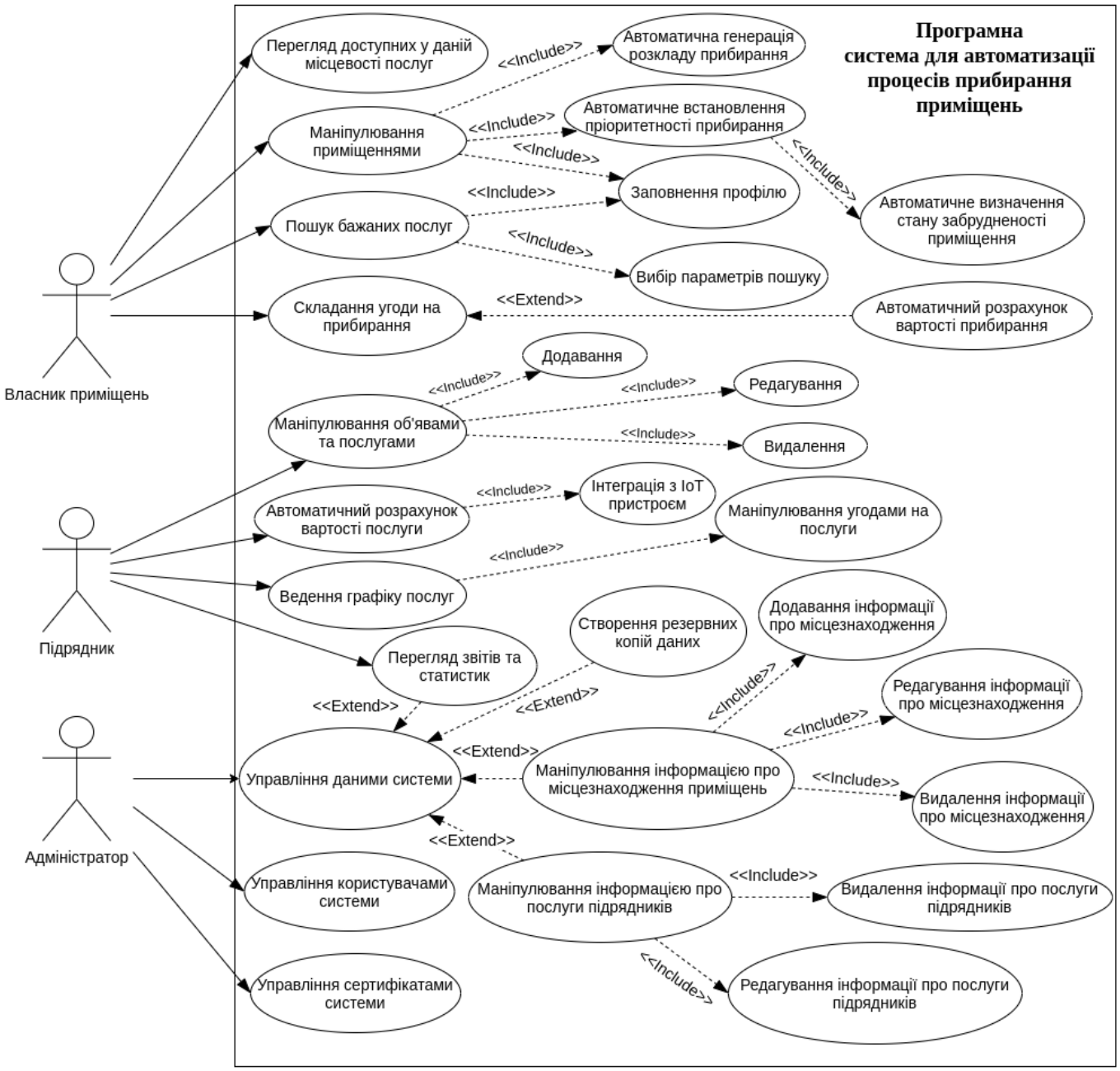

Рисунок А.1 – Діаграма варіантів використання для програмної системи автоматизації процесів прибирання приміщень «PurityService»

ДОДАТОК Б

Діаграма діяльності для мобільного застосунку

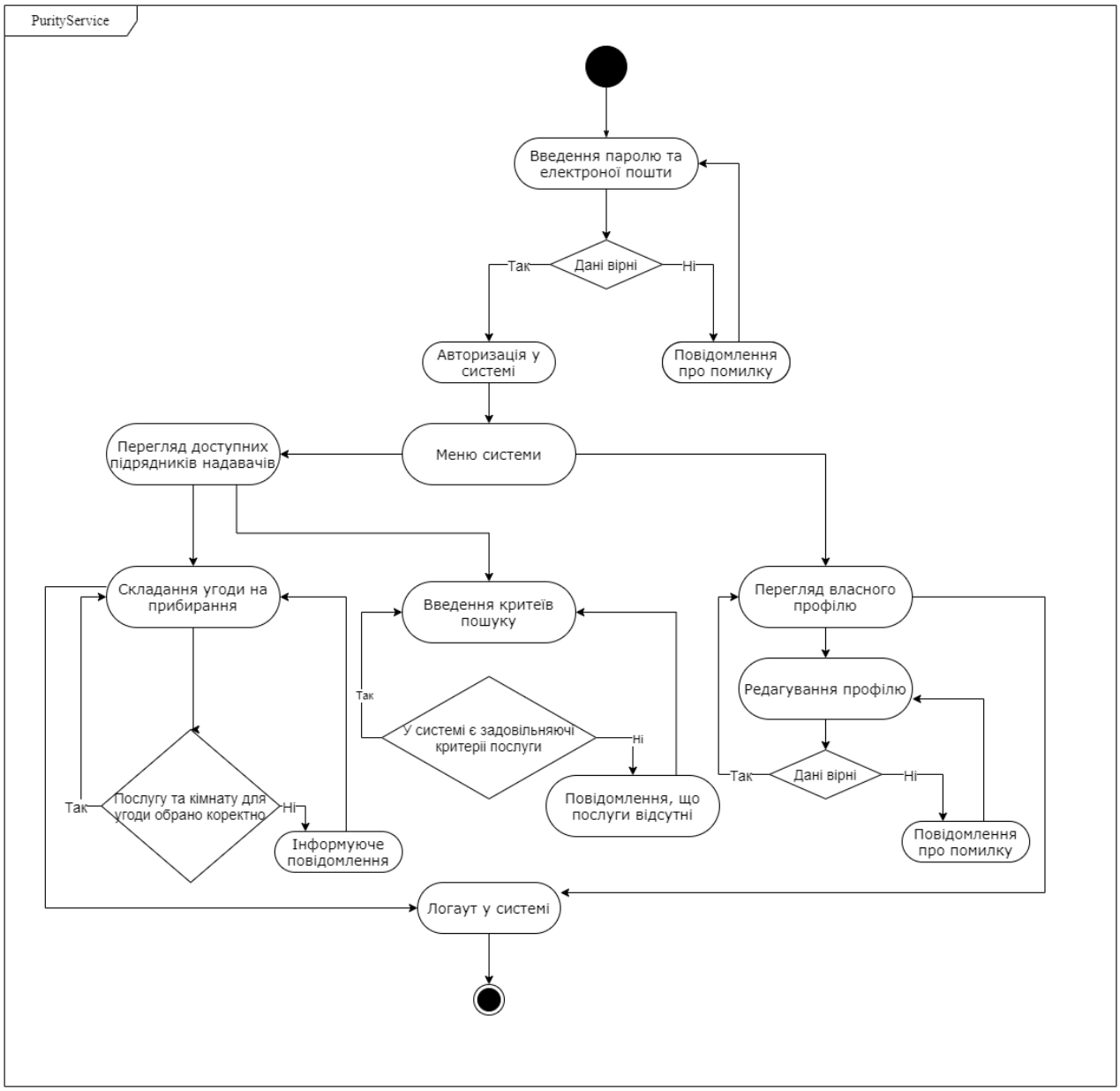

Рисунок Б.1 – Діаграма діяльності для мобільного застосунку програмної системи автоматизації процесів прибирання приміщень «PurityService»

ДОДАТОК В

Частина програмної реалізації запиту реєстрації користувача у системі

```
1   @Override
2   protected void onCreate(Bundle savedInstanceState) {
3       super.onCreate(savedInstanceState);
4       requestWindowFeature(Window.FEATURE_NO_TITLE);
5
6       getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
7               WindowManager.LayoutParams.FLAG_FULLSCREEN);
8       setContentView(R.layout.activity_sign_up);
9
10      context = this;
11
12      init();
13  }
14
15  private void signUp(String name, String email, String phone,
16                      String password, String role) {
17      if (!validateName(this, this.name)
18              || !validateEmail(this, this.email)
19              || !validatePhone(this, phoneNumber)
20              || !validatePassword(this, this.password)
21              || !validatePasswords(this, this.password,
22                    this.confirmPassword)) {
23          return;
24      } else if (!InternetConnection.checkConnection(
25                  getApplicationContext())) {
26
27          Toast.makeText(this,
28               R.string.no_internet_connection, Toast.LENGTH_LONG).show();
29
30          return;
31      } else {
32          User user = User.getInstance()
33                  .setName(name)
34                  .setEmail(email)
35                  .setPhoneNumber(phone)
36                  .setPassword(password)
37                  .setUserRole(role);
38
39          if (role.equals(getString(R.string.placement_owner))) {
40
41              NetworkService.getInstance()
42                      .getApiService()
43                      .placementOwnerSignUp(user)
44                      .enqueue(signUpCallback);
45
46          } else if (role.equals(getString(R.string.cleaning_provider))) {
47
48              NetworkService.getInstance()
49                      .getApiService()
50                      .cleaningProviderSignUp(user)
51                      .enqueue(signUpCallback);
52          }
53      }
54  }
```

```
55  private final Callback<User> signUpCallback = new Callback<User>() {
56      @Override
57      public void onResponse(Call<User> call, Response<User> response) {
58          if(!response.isSuccessful()) {
59              Log.i(TAG, response.message() + " " + response.code());
60
61              Toast.makeText(
62                      SignUpActivity.this,
63                      response.message(),
64                      Toast.LENGTH_SHORT
65              ).show();
66
67          } else {
68              Intent intent = new Intent(SignUpActivity.this,
69                      SignInActivity.class);
70              startActivity(intent);
71          }
72      }
73
74      @Override
75      public void onFailure(Call<Company> call, Throwable t) {
76          Log.i(TAG, t.getMessage());
77          Toast.makeText(
78                  SignUpActivity.this,
79                  t.getMessage(),
80                  Toast.LENGTH_SHORT
81          ).show();
82      }
83  };
```

ДОДАТОК Г

Частина програмної реалізації запиту на складання угоди на прибирання

```
1   private void getData() {
2       String email = User.getInstance().getEmail();
3       token = "Bearer " + User.getInstance().getToken();
4
5       mApi.getServiceData(token, getIntent().getStringExtra("cEmail"))
6               .enqueue(serviceCallback);
7       mApi.getPlacementData(token, email).enqueue(placementCallback);
8   }
9
10  Callback<ArrayList<Service>> serviceCallback =
11          new Callback<ArrayList<Service>>() {
12      @Override
13      public void onResponse(Call<ArrayList<Service>> call,
14              Response<ArrayList<Service>> response) {
15          if (!response.isSuccessful()) {
16              System.out.println(response.code());
17              return;
18          }
19          mServices = response.body();
20          String[] names = new String[mServices.size()];
21          servicesId = new int[mServices.size()];
22          int i = 0;
23          for (Service service : mServices) {
24              names[i] = service.getName();
25              servicesId[i] = service.getId();
26              i++;
27          }
28          initServicesAdapter(names);
29      }
30
31      @Override
32      public void onFailure(Call<ArrayList<Service>> call, Throwable t) {
33          System.out.println(t);
34          Log.i(TAG, t.getMessage());
35      }
36  };
37
38  Callback<ArrayList<Placement>> placementCallback =
39          new Callback<ArrayList<Placement>>() {
40      @Override
41      public void onResponse(Call<ArrayList<Placement>> call,
42              Response<ArrayList<Placement>> response) {
43          if (!response.isSuccessful()) {
44              System.out.println(response.code());
45              return;
46          }
47          mPlacements = response.body();
48          Integer[] ids = new Integer[mPlacements.size()];
49          int i = 0;
50          for (Placement placement : mPlacements) {
51              ids[i] = placement.getId();
52              i++;
53          }
54          initRoomsAdapter(ids);
```

```
55      }
56
57      @Override
58      public void onFailure(Call<ArrayList<Placement>> call, Throwable t) {
59          System.out.println(t);
60          Log.i(TAG, t.getMessage());
61      }
62  };
63
64  private void signContract() {
65      mContract.setProviderServiceId(servicesId[serviceId]);
66      mContract.setPlacementId(placementId);
67      SimpleDateFormat dateFormat = new SimpleDateFormat(
68              "yyyy-MM-dd'T'HH:mm:ss.SSSX");
69      mContract.setDate(dateFormat.format(new Date()));
70
71      NetworkService.getInstance().getApiService()
72              .signContract(token, mContract).enqueue(signContractCallback);
73  }
74
75  Callback<Contract> signContractCallback = new Callback<Contract>() {
76      @Override
77      public void onResponse(Call<Contract> call,
78              Response<Contract> response) {
79          if (!response.isSuccessful()) {
80              System.out.println(response.code());
81              return;
82          }
83          finish();
84      }
85
86      @Override
87      public void onFailure(Call<Contract> call, Throwable t) {
88          System.out.println(t);
89          Log.i(TAG, t.getMessage());
90      }
91  };
```