

## מטלה 4 – שלב הבדיקות

חברי הצוות: רעות גרבר 322516840, אור דרעי 211715453, שירת הוטרר 325445997

### חלק A:

#### 1. תרחיש ראשון : הרשמה לאירוע

##### Preconditions:

1. המשתמש מחובר לחשבון מסוג User
2. שירותי מיקום מופעלים
3. קיים אירוע זמין בסביבה של המשתמש

##### Steps:

1. המשתמש פותח את האפליקציה ונכנס אל למסך אירועים.
2. עובר לרשימת האירועים או למפה.
3. מבצע סינון לפי תאריך או מיקום.
4. בוחר אירוע עם מקום פנוי.
5. לוחץ על כפתור הירשם.

##### Expected Result:

מופיעה למשתמש הודעה שהרשמתו לאירוע הצליחה, והאירוע מתווסף לרשימת האירועים אליהם המשתמש רשום.

#### 2. תרחיש שני : יצירת אירוע

##### Preconditions:

1. המשתמש מחובר לחשבון מסוג Owner
2. למשתמש יש מקום תקף שבו תתקיים הופעה חיה ומקום לקהל

##### Steps:

1. המשתמש פותח את האפליקציה ונכנס למסך של יצירת אירוע
2. ממלא את כל פרטי האירוע ומוסיף מיקום של האירוע
3. לוחץ על יצירת אירוע

##### Expected Result:

מופיעה למשתמש הודעה שיצירת האירוע הצליחה, והאירוע נוצר בהצלחה ומופיע ברשימת אירועים של המשתמש.

#### 3. תרחיש שלישי : ביטול הרשמה לאירוע

##### Preconditions:

1. המשתמש מחובר לחשבון מסוג User
2. המשתמש רשום לאירוע שעדיין לא התקיים

##### Steps:

1. המשתמש פותח את האפליקציה ונכנס אל המסך של האירועים שהוא רשום אליהם
2. בוחר אירוע שאליו הוא מעוניין לבטל הרשמה
3. לוחץ על ביטול הרשמה

##### Expected Result:

מופיעה הודעה למשתמש שהרשמתו לאירוע בוטלה, והאירוע לא מופיע יותר במסך של האירועים שהמשתמש נרשם אליהם.

## חלק B:

בחלק זה כתבנו בדיקות יחידה למחלקה `CreateNewEventViewModel`, במטרה לבדוק את הלוגיקה של יצירת אירוע בצורה מבודדת וללא תלות בממשק המשתמש או בדאטה-בייס חיצוני. כתבנו 3 בדיקות, כאשר כל בדיקה בנויה לפי עקרון: Arrange - Act - Assert. **Arrange** - הכנת תנאי ההתחלה והנתונים לבדיקה, כולל שימוש ב-Mocks כשצריך. **Act** - קריאה לפונקציה אותה בודקים. **Assert** - בדיקה שהתוצאה שהתקבלה היא זו שמצופה.

### הבדיקות כוללות:

1. בדיקה של קלט לא תקין (לדוגמה: שם אירוע ריק).
2. בדיקה של לוגיקה פנימית לניהול ז'אנרים.
3. בדיקה של תרחיש תקין, כולל בדיקה שהאירוע נוצר עם הנתונים הנכונים ונשלח לריפוזיטורי.

צילומי מסך של קודי הבדיקות:

1.

```
/**
 * Test 1:
 * Publishing an event with an empty title should set TITLE error
 */
@Test
public void publish_emptyTitle_setsTitleError() {
    // Arrange
    viewModel.onLocationSelected( lat: 32.0, lng: 34.8, address: "Tel Aviv");
    viewModel.setDate( year: 2026, month: 0, day: 10);
    viewModel.setTime( hour: 20, minute: 0);
    viewModel.toggleGenre( genre: "Rock", checked: true);

    // Act
    viewModel.publish( name: "", capacity: "100", description: "Test description");

    // Assert
    assertEquals(EventField.TITLE, viewModel.getErrorField().getValue());
}
```

```
/**
 * Test 2:
 * getCheckedGenres should return a correct boolean array
 * according to the selected genres
 */

@Test
public void getCheckedGenres_returnsCorrectCheckedArray() {
    // Arrange
    String[] allGenres = {"Rock", "Jazz", "Pop"};

    viewModel.toggleGenre( genre: "Rock", checked: true);
    viewModel.toggleGenre( genre: "Pop", checked: true);

    // Act
    boolean[] checked = viewModel.getCheckedGenres(allGenres);

    // Assert
    assertTrue(checked[0]); // Rock selected
    assertFalse(checked[1]); // Jazz not selected
    assertTrue(checked[2]); // Pop selected
}
```

```

85      /**
86       * Test 3 (with Mock):
87       * Valid publish should call repository and succeed
88       */
89      @Test
90      public void publish_validEvent_callsRepositoryAndSucceeds() {
91          // Arrange (Mocking dependencies):
92          // Mock an asynchronous Firebase Task so the test is independent of Firebase
93          Task<Void> mockTask = mock(Task.class);
94
95          // Configure the repository to return the mocked Task when creating an event
96          when(mockEvent.createEvent(any(Event.class))).thenReturn(mockTask);
97
98          // Simulate successful and failure callback chaining on the Task
99          when(mockTask.addOnSuccessListener(any())).thenReturn(mockTask);
100         when(mockTask.addOnFailureListener(any())).thenReturn(mockTask);
101
102         // Arrange
103         viewModel.onLocationSelected( lat: 32.0, lng: 34.8, address: "Tel Aviv");
104         viewModel.setDate( year: 2026, month: 0, day: 10);
105         viewModel.setTime( hour: 20, minute: 0);
106         viewModel.toggleGenre( genre: "Rock", checked: true);

```

```

107
108         // Act
109         viewModel.publish( name: "Live Concert", capacity: "150", description: "Great music night");
110
111         // Assert
112         ArgumentCaptor<Event> captor = ArgumentCaptor.forClass(Event.class);
113
114         verify(mockEvent, times( wantedNumberOfInvocations: 1)).createEvent(captor.capture());
115
116         Event createdEvent = captor.getValue();
117
118         assertEquals( expected: "Live Concert", createdEvent.getName());
119         assertEquals( expected: 150, createdEvent.getMaxCapacity());
120         assertEquals( expected: "owner123", createdEvent.getOwnerId());
121
122         assertNull(viewModel.getErrorField().getValue());

```

## חלק C:

בחלק זה מימשנו בדיקות UI עבור מסך Create New Event באמצעות ספריית Espresso. מטרת הבדיקות היא לוודא את התנהגות ממשק המשתמש בעת אינטראקציה עם רכיבי UI שונים, וכן לאמת את מצב הממשק לאחר פעולות המשתמש (הצלחה, הצגת שגיאות ולידציה וניקוי שגיאות).

לצורך בדיקות יציבות וללא תלות ב-Firebase או רשת, ה-Activity מוזרק עם CreateNewEventViewModel שמשתמש בפייק-ריפוזיטורי.

בקוד נעשה שימוש בהלפרים פנימיים כגון מילוי שדות, בחירת תאריך/שעה/ז'אנר, ולחיצה על כפתור publish, כדי לשפר קריאות ולמנוע כפילויות.

בנוסף השתמשנו גם בהלפרים חיצוניים:

DialogHelpers.clickPositive - לחיצה יציבה על כפתור אישור בדיאלוגים.

EditTextAssertions.hasNoError - אימות שהוסרה לחלוטין שגיאת ולידציה משדה קלט.

LocationTestData - נתוני מיקום קבועים לצורך סימולציית בחירת מיקום.

### הבדיקות שבוצעו:

1. יצירת אירוע בהצלחה - מילוי כל השדות הנדרשים ולחיצה על Publish, עם אימות שה-Activity נסגר.

2. ולידציית שם אירוע - שליחת טופס ריק מציגה שגיאה, ותיקון הקלט מסיר את השגיאה.

צילומי מסך של הבדיקות:

פירוט ההלפרים הפנימיים:

```

52     private void fillName(String name) {
53         onView(withId(R.id.eventNameInput))
54             .perform(replaceText(name), closeSoftKeyboard());
55     }
56
57     2 usages
58     private void fillCapacity(String capacity) {
59         onView(withId(R.id.eventCapacityInput))
60             .perform(replaceText(capacity), closeSoftKeyboard());
61     }
62
63     1 usage
64     private void fillDescription(String description) {
65         onView(withId(R.id.eventDescriptionInput))
66             .perform(replaceText(description), closeSoftKeyboard());
67     }
68
69     2 usages
70     private void simulateLocationSelection(LocationTestData.LocationData location) {
71         rule.getScenario().onActivity( CreateNewEventActivity activity ->
72             activity.getViewModel().onLocationSelected(
73                 location.lat,
74                 location.lng,
75                 location.address
76             )
77         );
78     }
79
80     private void pickDate() {
81         onView(withId(R.id.dateInput)).perform(click());
82         clickPositive();
83     }
84
85     2 usages
86     private void pickTime() {
87         onView(withId(R.id.timeInput)).perform(click());
88         clickPositive();
89     }
90
91     2 usages
92     private void pickGenre(String genreText) {
93         onView(withId(R.id.genreSpinner)).perform(click());
94         onView(withText(genreText)).perform(click());
95         clickPositive();
96     }
97
98     3 usages
99     private void clickPublish() {
100         onView(withId(R.id.publishEventBtn)).perform(click());
101     }

```

## הטסטים: 1. יצירת אירוע בהצלחה

```
97      /**
98       * UI Test 1:
99       * Filling all required fields and clicking publish should complete the flow successfully.
100      */
101      @Test
102      public void createEvent_success_activityFinishes() {
103
104          // Arrange
105          injectTestingViewModel(
106              new FakeAuthRepository( uid: "test-owner-id"),
107              new FakeEventRepository().succeed()
108          );
109
110
111          // Act
112          simulateLocationSelection(LocationTestData.TEL_AVIV);
113          fillName("Jam Night");
114          pickDate();
115          pickTime();
116          pickGenre( genreText: "Rock");
117          fillCapacity("80");
118          fillDescription("Live music event");
119          clickPublish();
120
121          // Assert
122          rule.getScenario().onActivity( CreateNewEventActivity activity ->
123              org.junit.Assert.assertTrue(activity.isFinishing())
124          );
125      }
```

## 2. ביסיון שליחת טופס ריק, הצגת שגיאה, וניקוי השגיאה ע"י הכנסת קלט תקין.

```
128      * UI Test 2:
129      * Submitting an empty form should show a validation error,
130      * and fixing the input should clear the error.
131      */
132      @Test
133      public void createEvent_emptyThenFixName_errorAppearsAndClears() {
134
135          // Arrange
136          injectTestingViewModel(
137              new FakeAuthRepository( uid: "test-owner-id"),
138              new FakeEventRepository().succeed());
139
140          // Act 1: submit empty form
141          clickPublish();
142
143          // Assert 1: name validation error appears
144          onView(withId(R.id.eventNameInput))
145              .check(matches(
146                  androidx.test.espresso.matcher.ViewMatchers
147                      .hasErrorText( expectedError: "נא להזין שם אירוע" )
148              ));
149
150          // Act 2: user fixes the name
151          fillName("Jam Night");
152
153          // Assert 2: error is fully cleared (no icon, no message)
154          onView(withId(R.id.eventNameInput))
155              .check(hasNoError());
156      }
```

## חלק D:

\*לא היה ברור בדרישות המטלה האם לצרף את הדוחות שכתבנו או את הדוחות שכתבו על הקוד שלנו, אז צירפנו את שניהם.

### הדוחות שנכתבו על הקוד שלנו:

#### דוח 1:

ת"ז של מגיש הדוח: 315489534

Code review:

קבצים רלוונטיים: `CreateNewEventActivity` , `CreateNewEventViewModel`

תפקיד בפרוייקט: המודול אחראי על יצירת אירוע חדש במערכת. הוא מבוסס על ארכיטקטורת MVVM המודרנית:

`CreateNewEventActivity` (View) אחראית על התצוגה, איסוף הקלט מהמשתמש (תאריך, שעה, מיקום) והצגת שגיאות.

`CreateNewEventViewModel` ה"מוח" שמנהל את הנתונים, מבצע וולידציה (אימות) של השדות ומתקשר עם ה `Repositories` לשמירה ב `Databases`

נקודות לשימור:

- הפרדת רשויות: המעבר ל `ViewModel` נקי. ה `Activity` לא מחשבת כלום, היא רק "צופה" (`Observe`) בשינויים.
- הקוד כתוב בצורה קריאה, משתנים קרואים בשמות המאפיינים את מטרותם ונראות נקייה.
- ניהול שגיאות חכם: שימוש ב `Enum (EventField)` להעברת שגיאות הוא מקצועי מאוד ומונע טעויות הקלדה.
- שימוש ב: `Utility` הוצאת עיבוד התאריכים ל `DateUtils` שומרת על ה `ViewModel` ממוקד בלוגיקה עסקית בלבד.
- בטיחות נתונים: יצירת עותק חדש של רשימת הג'אנרים (`new ArrayList<>(genres)`) בעת יצירת האירוע מונעת באגים של סנכרון נתונים.

הערות מרכזיות לשיפור:

א. שיפור מבנה הקוד

- בתוך `setupListeners`, יש שני מאזינים (לכפתור המפה ולשדה הטקסט של המיקום) שמבצעים בדיוק את אותה פעולה: פתיחת ה-`MapPickerActivity`. אם נרצה לשנות בעתיד את ה-`Intent` או להוסיף פרמטר, תצטרך לעדכן בשני מקומות.

מומלץ ליצור פונקציה קטנה בשם `openMapPicker()` ולקרוא לה משני המקומות.

ב.

- פונקציית ה `publish` ב `view modle` מבצעת גם וולידציה וגם שמירה. מומלץ לפצל אותה כדי לשפר את הקריאות (למשל, ליצור פונקציה פרטית `isValidInput()`).



ג. ניהול מחזוריות (Coding Standards)

- הודעות שגיאה (כמו "נא להזין שם אירוע") והכותרות בדיאלוגים כתובות כטקסט חופשי (Hardcoded). יש להעביר אותן לקובץ `res/values/strings.xml` לאפשר תמיכה בעברית/אנגלית בצורה תקנית.

דוגמה לשיפור עבור ב:

ניצור פונקציה בשם `isValid` שתבדוק בצורה פשוטה האם ה-`INPUT` שקיבלנו תקין ותחזיר `TRUE` או `FALSE` בהתאם למה שיוחזר תפעל הפונקציה `PUBLISH`

## דוח 2:

קובץ שנבדק: `EditEventActivity.java`, `EditEventViewModel.java`

ת"ז הבודק: 332520626

### קריאות הקוד

הקוד מציג רמת קריאות טובה וארגון ברור. שמות המתודות כמו: `observeViewModel`, `initViews`, `setUpListeners`, מתארים היטב את מטרתן ומקלים על הבנת זרימת הקוד. שמות המשתנים ברובם ברורים וכתובים בדרכים המקובלים לפיתוח בסביבת אנדרואיד.

עם זאת, ניתן לשפר מספר דברים כמו: שם המשתנה "c" אינו ברור ויש להחליף אותו בשם יותר מובן כמו "calendar". בנוסף, שמות של רכיבי UI מסוימים (כגון `btnMap`) יכולים להיות מפורטים יותר (למשל: `btnOpenMap`). יש מתודות שלא כל כך ברור מה מטרתן, לכן מאוד יעזור להוסיף הערות עם הסבר קצר על מתודות.

### שיפור מבנה הקוד

הקוד בנוי היטב ומחולק למתודות קטנות ובעלות אחריות ברורה. השימוש ב-`ViewModel` מפריד בין הלוגיקה לבין ה-UI, בהתאם לעקרונות ארכיטקטורת MVVM. המחלקה `SimpleTextWatcher` היא דוגמה טובה לצמצום קוד חוזר וליישום עקרון DRY.

עם זאת, קיימת עדיין חזרתיות במקומות מסוימים כמו: ניקוי שגיאות ולידציה מתבצע על מספר רכיבי UI בנפרד, אפשר לרכז את ניקוי השגיאות במתודת עזר אחת. יש מספר רכיבים מפעילים את אותה פעולה (פתיחת בחירת מיקום במפה), אך הלוגיקה משוכפלת במקום להיות ממוחזרת. המתודה `observeViewModel` מטפלת במספר רב של אחריות ואפשר לפצל אותה לתת-מתודות לצורך קריאות ותחזוקה טובים יותר.

## איתור באגים ועילות

קיימת בעיה פוטנציאלית בשימוש באובייקט Calendar. האובייקט מאותחל פעם אחת מתוך ה-ViewModel, אך אינו מתעדכן לאחר בחירת תאריך או שעה חדשים על ידי המשתמש. מצב זה עלול לגרום לפתיחת בוחרי תאריך או שעה עם ערכים לא מעודכנים.

מבחינת יעילות, לוגיקת טיפול בשגיאות יכולה להיות מרוכזת במקום אחד כדי לצמצם עדכוני UI מיותרים.

## תקני קידוד וקונבנציות

הקוד עומד ברוב תקני הקידוד של Java ו-Android: הזחה ועיצוב אחידים, שימוש נכון ב-access modifiers ושימוש נכון ברכיבים מודעי מחזור חיים כגון ViewModel ו-LiveData.

נקודות לשיפור:

- הימנעות ממחרוזות וקבועים "קסומים" בתוך הקוד.
- הוספת אנוטציות @NonNull ו-@Nullable לשיפור בדיקות סטטיות ומניעת שגיאות ריצה.
- תיעוד של ממשקים פנימיים ומתודות עזר.

## סיכום

### שלוש הערות מרכזיות

1. **שיפור קריאות ותיעוד:** שינוי שמות משתנים לא ברורים והוספת תיעוד למתודות עזר לשיפור הבנת הקוד.
2. **צמצום קוד חוזר:** ריכוז לוגיקה משוכפלת (כגון ניקוי שגיאות ופתיחת מפה) למתודות עזר בהתאם לעקרון DRY.
3. **תיקון בעיות מצב:** סנכרון אובייקט Calendar עם בחירות המשתמש למניעת חוסר עקביות בהתנהגות הממשק.

**דוגמא לשיפור:** במקום לכתוב את כל ניקוי השגיאות כל פעם, אפשר לכתוב מתודה אחת שיש בה את כל הניקויי שגיאות וכשצריך לנקות שגיאות אפשר לקרוא למתודה. כך הקוד יראה יותר נקי ואפשר גם למחזר קוד.

## דוח 3:

**Code Review - תז: 211602222**

**קבצים רלוונטיים** ExploreEventsActivity.java, ExploreEventsViewModel.java

**תפקיד בפרויקט :** המודול אחראי על הצגת אירועים על גבי מפה וסינונם לפי מיקום המשתמש. המודול מבוסס על ארכיטקטורת **MVVM** המפרידה בין שכבת התצוגה ללוגיקה העסקית:

- **ExploreEventsActivity (View)** - אחראית על ניהול רכיבי ה UI-הצגת המפה, כפתורי הסינון וצפייה (Observation) בשינויי המידע.

- **ExploreEventsViewModel** - משמש כ"מוח" המנהל את שליפת הנתונים מה Repository, מבצע חישובים גיאוגרפיים וניהול מצבי UI.

#### נקודות לשימור:

- **הפרדת רשויות**: קיימת הפרדה נקייה בין הלוגיקה לתצוגה. ה Activity אינה מבצעת חישובים אלא רק מציגה נתונים המגיעים מה ViewModel.
- **שימוש ב Utility**: הוצאת חישובי המרחק ל GeoUtils והשימוש ב MapUiHelper לצורך המפה שומרים על קוד מודולרי וקל לתחזוקה.
- **ניהול מצב (State Management)**: השימוש ב isEmpty LiveData לשליטה על נראות הודעת ה "No events found" - הוא מימוש תקני ומקצועי.
- **קריאות הקוד**: שמות המשתנים והפונקציות ברורים ומשקפים את מטרתם בצורה טובה.

#### הערות מרכזיות לשיפור:

א. **שיפור מבנה הקוד (DRY - Don't Repeat Yourself)**: בתוך ה ViewModel, הפונקציות loadEventsNearMe ו- loadAllEvents ל eventRepository.getActiveEvents(). מומלץ לאחד את שליפת הנתונים לפונקציה אחת המקבלת פרמטר סינון, כדי למנוע כפילות בקוד.

ב. **הפרדת לוגיקה לשיפור היעילות והבדיקות**: פונקציית הטעינה הנוכחית מבצעת גם שליפה מהרשת וגם סינון גיאוגרפי. מומלץ לפצל את פעולת הסינון לפונקציה פרטית נפרדת. הדבר משפר את קריאות הקוד ומאפשר לבצע בדיקות יחידה (Unit Tests) על לוגיקת המרחקים בקלות.

ג. **ניהול מחרוזות (Coding Standards)**: מחרוזות כגון "Hello" והודעות שגיאה כתובות כטקסט חופשי (Hardcoded) בתוך הקוד. יש להעבירן לקובץ res/values/strings.xml כדי לאפשר תמיכה בריבוי שפות (לוקליזציה) ושינוי טקסטים ללא נגיעה בקוד.

#### דוגמה לשיפור עבור סעיף ב'

יצירת פונקציה פרטית בשם filterByProximity המבודדת את הלוגיקה המתמטית מהקריאה לשרת. הפונקציה כוללת בדיקת תקינות (Validation) לקואורדינטות.

### הדוחות שכתבנו על קוד של קבוצה אחרת:

#### דוח 1:

תז: 325445997

הקבצים שנבדקו: ReportItemViewModel, ReportFoundItemActivity.

סוג הקוד - (MVVM) UI + ViewModel

תיאור כללי: הקוד שנבדק אחראי על תהליך דיווח פריט שנמצא באפליקציה. ה Activity מטפלת באינטראקציה עם המשתמש, כולל הזנת פרטי הפריט (קטגוריה, תיאור, מיקום ותאריך), טיפול בהרשאות מצלמה ובצילום תמונה. בנוסף, מתבצעות בה פעולות כמו המרת התמונה ל-Base64 וחישוב קואורדינטות גאוגרפיות מתוך כתובת טקסטואלית.

ה- ViewModel אחראי על ולידציה של שדות חובה, יצירת אובייקט FoundItem ושליחת הנתונים ל Repository לצורך שמירה. החלוקה בין ה Activity ל ViewModel משקפת שימוש בעקרונות MVVM, עם הפרדה ברורה יחסית בין ממשק המשתמש ללוגיקה העסקית.

### נקודות חיוביות בקוד:

- שימוש ברור ועקבי ב ViewModel לצורך בידוד לוגיקה מה UI.
- שמות משתנים ופונקציות קריאים וברורים.
- הערות ותיעוד איכותי ומובן שמסביר את מטרת הפונקציות.
- אינטראקציה נכונה בין השכבות (UI -> ViewModel -> Repository)

### הערות מרכזיות:

**1. הפרדת אחריות** - במהלך קריאת הקוד ניתן לראות שחלק מהלוגיקה שנמצאת ב Activity אינה קשורה ישירות ל UI, כמו המרת תמונה ל Base64 וחישוב קואורדינטות מתוך כתובת. פעולות אלו שייכות יותר ללוגיקה עסקית, והימצאותן ב Activity יוצרת עומס מיותר.

הצעה לשיפור - מומלץ להעביר פונקציות כמו `encodeImageToBase64` ו- `getLatLongFromAddress` ל ViewModel או למחלקות עזר ייעודיות. כך ה Activity תישאר אחראית בעיקר על קלט ותצוגה, והקוד יהיה ברור וקל יותר לתחזוקה.

**2. קריאות הקוד** - קיימת כפילות לוגית בין הפונקציות `reportFoundItem` ו- `reportLostItem`. בשתיהן מתבצעים שלבים דומים מאוד, כמו ולידציה, יצירת אובייקט ושליחה ל Repository.

הצעה לשיפור - כדי לשפר את הקריאות ולצמצם כפילויות, ניתן לאחד את הלוגיקה המשותפת לפונקציה כללית אחת, או להשתמש במבנה משותף ליצירת האובייקטים. שינוי כזה יקצר את הקוד ויקל על שינויים עתידיים.

**3. טיפול בשגיאות** - במקרים שבהם מתרחש כשל בתהליך (ולידציה, חישוב מיקום או שמירה ב Firestore), מוחזר ערך `false` בלבד, ללא פירוט. מצב זה מקשה על איתור בעיות וגם על מתן משוב ברור למשתמש.

הצעה לשיפור - מומלץ להחזיר מידע מפורט יותר על סיבת הכשל, להוסיף לוגים רלוונטיים, ולהציג למשתמש הודעה מתאימה במקרה של שגיאה. כך ניתן לשפר גם את חוויית השימוש וגם את תהליך הדיבוג.

### דוגמא לשיפור בעקבות Review:

לפני השיפור, ה Activity כללה לוגיקה של המרת תמונה וחישוב קואורדינטות. בהצעה לשיפור, לוגיקה זו מועברת ל ViewModel, וה Activity אחראית רק על איסוף הקלט מהמשתמש וקריאה לפונקציה אחת ב ViewModel.

שינוי זה מוביל לקוד פשוט יותר, עם חלוקה ברורה של אחריות, ומשפר את יכולת הבדיקות והתחזוקה של המערכת.

ת"ז-211715453

קבצים רלוונטים: LoginActivity

א. בזרימת ההתחברות עם Google קיימת אפשרות ש GoogleSignInAccount יחזור null במצבים מסוימים. במצב כזה קריאה ישירה ל-firebaseAuthWithGoogle עלולה לגרום לקריסה.

ב. יש כמה פונקציות כמו attemptLogin, performFirebaseLogin, firebaseAuthWithGoogle שחסר להם תיעוד וזה מקשה על קריאת הקוד, אפשר להוסיף אפילו משפט אחד שמתאר את המטרה של הפונקציה.

ג. LoginActivity כרגע עמוס מדי ומבצעת פעולות שלא קשורות לתצוגה, כמו תקשורת ישירה מול ה-API של Firebase. עדיף להעביר את כל הלוגיקה של Firebase ל-ViewModel-ה-Activity צריכה להיות אחראית אך ורק על הצגת הנתונים וקבלת קלט מהמשתמש.

דוגמה לשיפור:

להוסיף בדיקה שהמשתנה GoogleSignInAccount לא null.

### דוח:3

תז: 322516840

## Code Review Summary Document

### Module:

ReportLostItemActivity.java

### Purpose:

מסך דיווח על פריט אבוד, הכולל קליטת נתונים מהמשתמש, צילום תמונה, המרת תמונה ל Base64- המרת כתובת לקואורדינטות (Lat/Lng) ושליחת המידע ל ViewModel-לצורך שמירה/עיבוד.

### Key Comments:

#### 1. הפרדת אחריות בהתאם ל-MVVM-

כיום ה-Activity מבצע לוגיקה עסקית משמעותית (המרת תמונה ל-Base64, גיאוקודינג כתובת (Lat/Lng). לפי עקרונות MVVM, מומלץ להעביר את הלוגיקה הזו ל-ViewModel או Repository. ה-Activity אמור להתמקד בקלט/תצוגה בלבד (UI Layer), בעוד שכבת ה-ViewModel אחראית על עיבוד הנתונים.

מומלץ לצורך שיפור מודולריות, קלות תחזוקה, בדיקות (Unit Tests) פשוטות יותר והקטנת עומס על שכבת UI.

#### 2. ניהול משאבים ויציבות

בפונקציה encodeImageToBase64() נפתח InputStream אך אינו נסגר בצורה מפורשת. זה מהווה דליפת משאבים (Resource Leak) ועלול להוביל לבעיות יציבות לאורך זמן.

בנוסף, פעולות כבדות כמו Base64 ו Bitmap processing-עלולות לגרום לעומס זיכרון ולירידה בביצועים.

המלצה: שימוש ב try-with-resources והעברת העיבוד לרקע. (Background Thread)

### 3. טיפול בשגיאות וחווית משתמש

חסרה ולידציה בסיסית לשדות חובה (כגון Category/Description/Date). כמו כן, קריאת Geocoder היא פעולה Blocking שעלולה להקפיא את ה-UI בזמן submitForm().

המלצה:

- ולידציה לפני שליחה (עם הודעות שגיאה ספציפיות).
  - הצגת Loading Indicator/Spinner בזמן עיבוד ושליחה.
  - הרצת Geocoder/עיבוד תמונה ברקע.
- השפעה: פחות "תקיעות", פחות דיווחים כפולים, וחווית משתמש מקצועית יותר.

### **Example of Improvement:**

הועברה לוגיקת Base64 + Geocoding (ל ViewModel-או, Repository) ונוסף שימוש ב try-with-resources לסגירת זרמי נתונים.  
לפני השינוי: כל העיבוד התבצע ב submitForm() בתוך ה Activity-כולל גיאוקודינג וקידוד Base64.  
לאחר השינוי: Activity-רק אוסף נתונים גולמיים ומוסר ל ViewModel-שמבצע את העבודה הכבדה ברקע.  
כתוצאה מכך הקוד נהיה קריא, ניתן לתחזוקה ובנוי בצורה נכונה מבחינת שכבות הארכיטקטורה.

### **Code Review Summary Document**

#### **Module:**

ReportItemViewModel.java

#### **Purpose:**

ViewModel שאחראי על הלוגיקה העסקית בדיווח פריטים אבודים/נמצאים: ולידציה לשדות חובה, בניית מודלים (LostItem / FoundItem) והאצלת פעולת העלאה ל Repository (Firestore/Storage)-באמצעות callback.

#### **Key Comments:**

#### מודולריות והפחתת כפילויות

יש כפילות בין reportLostItem ל: reportFoundItem-  
שתי הפונקציות מבצעות את אותם שלבים כמעט בדיוק (ולידציה - יצירת מודל- שדות set-קריאה ל repository).  
מומלץ לחלץ את הלוגיקה המשותפת לפונקציה/Builder משותף, או לחלופין להשתמש בפולימורפיזם (למשל באמצעות ממשק משותף ל FoundItem ו-LostItem, או תבנית עיצוב מסוג Factory/Mapper).

באופן פרקטי ניתן ליצור מתודה פרטית שמקבלת את "סוג הדיווח" (אבוד/נמצא) ומחזירה אובייקט מוכן, או ליצור ItemFactory / Mapper שאחראיים לבניית המודל עצמו, בעוד שה- ViewModel יתמקד רק בניהול הזרימה ותזמון הקריאות ל-Repository.

וולידציה ובדיקות

הוולידציה `isValid()` בודקת רק אם הערכים הם null או חסרים מקרים נפוצים שעלולים לגרום לשגיאות: למשל מחרזות שמכילות רק רווחים עדיין ייחשבו כ"לא ריקות", ייתכן שה- `callback` יגיע כ- null או קריאה ל- `callback.accept(...)` תגרום לקריסה `imageUri`, יכול להיות null תוקן אך `base64Image` לעיתים עשוי להיות null או `setImageBase64` עלול לגרום ל- `NPE` (תלוי במימוש של המודל), ובנוסף ערכי `lat/lng` מתמיד מועברים אך במקרה שה- `Geocoder` נכשל הפעילות שולחת `0.0/0.0` זוהי נראה "תקין" למרות שמדובר במיקום לא אמיתי ולכן כדאי להגדיר מדיניות ברורה האם לאפשר ערך זה או להחזיר כישלון/אזהרה; לכן מומלץ להשתמש ב- `trim()` בדיקות הריקות, להגן מפני מצב שבו `callback` הוא null, ולהגדיר כללי תקינות לקואורדינטות (למשל `boolean` `hasCoordinates` או שימוש בערכים, nullable), מה שיקטין קריסות וימנע שמירה של דיווחים שגויים עם מיקום לא אמין.

## ניהול תלויות ותשתיות

במחלקה הנוכחית ה- `ViewModel` יוצר את ה- `Repository` בתוך עצמו באמצעות `private final`;  
`repository = new ItemRepository();`  
 (במיוחד `Mocking` ופוגע בעיקרון של `Dependency Injection` משום שלא ניתן להחליף את ה-  
`Repository` במימוש חלופי לצורך טסטים או הרחבה; בנוסף, השימוש ב- `<Boolean>Consumer` הוא  
 מנגנון `callback` של `Java 8` אך בפיתוח `Android` מודרני נהוג לעבוד עם `callbacks` מותאמים או עם  
 מנגנונים כמו `LiveData / StateFlow` שמספקים זרימה מסודרת של מצב למסך, ולכן מומלץ להזריק את  
 ה- `Repository` דרך ה- `constructor` (באמצעות `ViewModel Factory` או לפחות לאפשר  
`constructor/package-private setter` למקביל לשקול להחזיר תוצאות דרך  
`<Result>LiveData` או אובייקט `State` כדי שה- `UI` יוכל להציג טעינה/שגיאה/הצלחה בצורה עקבית  
 וסטנדרטית.

### Example of Improvement:

השינוי שבוצע כולל חיזוק מנגנון הוולידציה והגנה מפני מצב שבו `callback` הוא `null`, תיקון מקרה של שדות שמכילים רווחים בלבד. לפני השיפור, הפונקציה `isValid()` בדקה רק `null/empty`, תולכן מחרוזות כמו " " עברו כקלט תקין, ובנוסף אם ה-`callback` הגיע כ-`null` מקריאה ל-`callback.accept(...)` התייתה גורמת לקריסה. לאחר השיפור, מבצעים `trim()` לכל שדה טקסט לפני בדיקות הוולידציה ומוסיפים בדיקת `guard` שמוודאת שה-`callback` אינו `null` לפני השימוש בו. כתוצאה מכך מתקבל קוד יציב יותר, עם פחות קריסות, ולידציה נכונה שמונעת שמירת דיווחים "ריקים" בפועל.