

מטרה 4 – שלב הבדיקה

חברי הצוות: רעות גרביר 0322516840, אור דרעי 3211715453, שירות הוטר 325445997

חלק A:

1. תרחיש ראשון : הרשמה לאירוע

Preconditions:

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

Steps:

1. המשתמש פותח את האפליקציה ונכנס אל מסך אירועי.
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:

מופיעו למשתמש הודעה שהרשמה לאירוע הצלחה, והאירוע מתווסף לרשימת האירועים אליום המשתמש רשום.

2. תרחיש שני : יצירת אירוע

Preconditions:

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

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:

מופיעו למשתמש הודעה שיצירת האירוע הצלחה, והאירוע נוצר בהצלחה ומופיע ברשימה של האירועים 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 אליום
2. בוחר אירוע אליו הוא מעוניין לבטל הרשמה
3. לוחץ על ביטול הרשמה

Expected Result:

מופייע הودעה למשתמש שהרשמה לאירוע בוטלה, והאירוע לא מוופיע יותר במסך של האירועים שהמשתמש נרשם אליהם.

חלק B:

ב חלק זה כתבנו בדיקות יחידה למחלקה `CreateNewEventViewModel`, במטרה לבדוק את הלוגיקה של יצירת אירוע בצורה מבודדת ולא תלות במסמך המשתמש או בDATA-BEAN חיצוני.

כתבנו 3 בדיקות, כאשר כל בדיקה בנויה לפי עקרון: Arrange - Act - Assert:

- **Arrange** - הכנות תנאי ההתחלה והנתונים לבדיקה, כולל שימוש ב-Mocks כנדרש.

- **Act** - קרייה לפונקציה אותה בודקים.

- **Assert** - בדיקה שההתוצאה שהיא קיבלת היא זו שמצופה.

בדיקות 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 ונסלח לריפזיטורי.

צילומי מסך של קוד הבדיקות:

.1

```
/*
 * Test 1:
 * Publishing an event with an empty title should set TITLE error
 */
@Test
public void publish_emptyTitle_setsTitleError() {
    // Arrange
    viewModel.setLocationSelected( 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());
}
```

.2

```
/*
 * 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.setLocationSelected( 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:

ב חלק זה מימושנו בבדיקות UI עבור מסך Create New Event באמצעות ספריית Espresso. מטרת הבדיקות היא לוודא את התנהלות משקל המשתמש בעת אינטראקציה עם רכיבי UI שונים, וכן לאמת את מצב המשקל לאחר פעולות המשתמש (הצלחה, הצגת שגיאות ולידציה ונקיי שגיאות).

לצורך בדיקות יציבות ולא תלות Firebase או רשות, ה Activity מזמין עם CreateNewEventViewModel שימוש המשתמש בפיק-ריפזיטורי.

בקוד נעשה שימוש בהלפרים פנימיים כגון مليוי שדות, בחירת תאריך/שעה/אזור, ולהיצה על כפטור publish, כדי לשפר קריונות ומניע כפיליות.

בנוסף השתמשנו גם בהלפרים חיצוניים:

DialogHelpers.clickPositive - להיצה יציבה על כפטור אישור בדיאלוגים.

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

LocationTestData - נתוני מקום קבועים לצורך סימולציה בחירת מקום.

בדיקות שביצעו:

1. יצירת אירוע בהצלחה - مليוי כל השדות הנדרשים ולהיצה על Publish, עם אימות שהעיה נסגר.

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

犹如我所写的检查：

ピロット ハルフリム 内部：

```
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     private void pickDate() {
80         onView(withId(R.id.dateInput)).perform(click());
81         clickPositive();
82     }
83
84     2 usages
85     private void pickTime() {
86         onView(withId(R.id.timeInput)).perform(click());
87         clickPositive();
88     }
89
90     2 usages
91     private void pickGenre(String genreText) {
92         onView(withId(R.id.genreSpinner)).perform(click());
93         onView(withText(genreText)).perform(click());
94         clickPositive();
95     }
96
97     3 usages
98     private void clickPublish() {
99         onView(withId(R.id.publishEventBtn)).perform(click());
100    }
```

הטוטוים:
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. ניסיון 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: "לא נמצא 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 יהיה ברור בדרישות המطلיה האם לצרף את הדוחות שכתבנו או את הדוחות שכתבו על הקוד שלנו,
אץ צירפנו את שניהם.

הדוחות שנכתבו על הקוד שלנו:

דוח 1:

ת"ז של מגיש הדוח: 315489534

Code review:

קבצים רלוונטיים: CreateNewEventActivity , CreateNewEventViewModel

תפקיד בפרויקט: המודול אחראי על ייצור אירוע חדש במערכת. הוא מבוסס על ארכיטקטורת MVVM המודרנית:

(View) CreateNewEventActivity אחראית על התצוגה, איסוף הקלט מהמשתמש (תאך, שעה, מיקום) והציגת שגיאות.

CreateNewEventViewModel מוח "מנהל" את הנתונים, מבצע וlidziah (אימות) של השדות Database לשימירה Repositories וمتקשר עם ה

נקודות לשימור:

- הפרדת רשויות: המעביר ל ViewModel נקי. ה Activity לא מחשבת כלום, היא רק "צופה" (Observe) בשינויים.
- הקוד כתוב בצורה קריאה, משתנים קרואים בשמות המאפיינים את מטרתם ונראות נקייה.
- ניהול שגיאות חכם: שימוש ב (EventField)Enum להעברת שגיאות הוא מקצועית מאוד ומונע טעויות הקלה.
- שימוש ב Utility: עיבוד התאריכים ל DateUtils שומרת על ה ViewModel ממוקד בלוגיקה עסקית בלבד.
- בתייחות נתונים: ייצור עותק חדש של רשימת הגאנרים (new ArrayList<genres>) בעת ייצור האירוע מונעת באגים של סנכרון נתונים.

הערות מרכזיות לשיפור:

- א. שיפור מבנה הקוד
- בטור setupListeners, יש שני מאזינים (לכפתור המפה ולשדה הטקסט של המיקום) שמבצעים בדיק את אותה פעולה: פТИחת-mapPickerActivity. אם נרצה לשנות בעידן Intent או להוסיף פרמטר, תצטרך לעדכן בשני מקומות.
 - מומלץ ליצור פונקציה קטנה בשם openMapPicker() ולקרא לה משני המקומות.
- .ב.
- פונקציית hash ב publish view model מבצעת גם וlidziah וגם שמירה. מומלץ לפצל אותה כדי לשפר את הקריאות (למשל, ליצור פונקציה פרטית. ()isValidInput))

ג. ניהול מחuzeות(Coding Standards)

- הודיעות שגיאה (כמו "נא להזין שם אירע") והכוורות בדיאלוגים כתובות בטקסט חופשי(Hardcoded) יש להעביר אותן לקובץ xml/values/strings.xml כדי לאפשר תמייה בעברית/אנגלית בצורה נכונה.

דוגמה לשיפור עבור ב:

ניצור פונקציה בשם `isValidInput` שתבדוק בצורה פשוטה האם ה`INPUT` שקיבלנו נכון ותחזיר TRUE או FALSE בהתאם למה שיוחזר תפעול 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 בדרך כלל מקובלות 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, لكن מאוד יעזור להוסיף הערות עם הסבר קצר על מתודות.

שיפור מבנה הקוד

הקוד בני הייטב ומוחלך למתחמות 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 ולישום עקרון DRY.

עם זאת, קיימת עדין חזרתיות במקומות מסוימים כמו: ניקוי שגיאות ולידציה מתבצע על מספר רכיבי UI בנפרד, אפשר לרכוץ את ניקוי השגיאות במתחמת עזר אחת. יש מספר רכיבים מפעילים את אותה פעולה (פתיחת בחירת מקום במפה), אך הלוגיקה משוכפלת במקום להיות מוחזרת. המתודה `observeViewModel` מטפלת במספר רב של אחריות ואפשר לפצל אותה לחת-מתודות לצורך קריאות ותחזקה טוביים יותר.

איתור באגים ויעילות

קיים בעה פוטנציאלית בשימוש באובייקט `Calendar`. האובייקט מאוחחל פעם אחת מתוך ה-`ViewModel`, אך אין מתעדכן לאחר בחירת תאריך או שעה חדשות על ידי המשתמש. מצב זה עלול לגרום לפתיחת בוחר תאריך או שעה עם ערכים לא מעודכנים.

מבחינת יעילות, לוגיקת טיפול בשגיאות יכולה להיות מרכזת במקום אחד כדי לצמצם עדכונו או מיותרים.

תקני קידוד וקונבנציות

הקוד עומד ברוב תקני הקידוד של Java ו-`Android`: הזהה ועיצוב אחידים, שימוש נכון ב-`modifiers`, שימוש נכון ב-`LiveData` ו-`ViewModel`.

נקודות לשיפור:

- הימנעות מחזרות וקבועים "קסומים" בתוך הקוד.
- הוספת אנטציות `@NotNull` ו-`@NonNull` לשיפור בדיקות סטטיות ומינעת שגיאות ריצה.
- תיעוד של ממשקים פנימיים וمتודות עזר.

סיכום

שלוש הערות מרכזיות

1. **שיפור קריאות ותיעוד:** שינוי שמות משתנים לא בוררים והוספה תייעוד למתחדות עזר לשיפור הבנת הקוד.
2. **צמצום קוד חוזר:** ריכוז לוגיקה משוכפלת (כגון ניקוי שגיאות ופтиחת מפה) למתחודות עזר בהתאם לעקרון DRY.
3. **תיקון בעיות מצב:** סyncron אובייקט `Calendar` עם בחירות המשתמש למניעת חוסר עקבות בהתקשרות המשך.

דוגמא לשיפור: במקומות לכתוב את כל ניקוי השגיאות כל פעם, אפשר לכתוב מתחודה אחת שיש בה את כל הניקוי' שגיאות וכשצריך לנוקות שגיאות אפשר לקרוא למתחודה. כך הקוד יראה יותר נקי ואפשר גם למחזר קוד.

דוח 3: תז: 211602222 -Code Review

קבצים רלוונטיים `ExploreEventsActivity.java`, `ExploreEventsViewModel.java`

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

- **ExploreEventsActivity (View)** - אחראית על ניהול רכיבי ה-UI-הציגת המפה, כפתורי הסינון וצפיה (Observation) בשינוי המידיע.

- **ExploreEventsViewModel** - מושך כ"מוך" המנהל את שליפת הנתונים מה Repository, מבצע חישובים גיאוגרפיים וניהול מצבו הלא.

נקודות לשימור:

- **הפרדת רשות:** קיימת הפרדה נקייה בין הלוגיקה לתצוגה. ה Activity-ACTIVITY-אינה מבצעת חישובים אלא רק מציגה נתונים המגיעים מה ViewModel.
- **שימוש ב Utility:** הוצאה חישובי המרחק ל GeoUtils והשימוש ב MapUiHelper לציר המפה שומרים על קוד מודולרי וקל לתחזוקה.
- **ניהול מצב (State Management):** השימוש LiveData isEmpty לשליטה על נראות הודעה "No events found" הוא יימוש תקין ומקצועי.
- **קריאות הקוד :** שמות המשתנים והפונקציות ברורים ו邏輯יים את מטרתם بصورة טובה.

הערות מרכזיות לשיפור:

- **שיפור מבנה הקוד (DRY - Don't Repeat Yourself):** בהתאם loadAllEvents loadEventsNearMe הפונקציות משכפלות את הקריאה eventRepository.getActiveEvents()(). מומלץ לאחד את שליפת הנתונים לפונקציה אחת המתקבלת פרמטר סינון, כדי למנוע כפילות בקוד.
- **הפרדת לוגיקה לשיפור הייעילות והבדיקות:** פונקציית הטעינה הנוכחיית מבצעת גם שליפה מהרשת וגם סינון גיאוגרפי. מומלץ לפחות את פעולה הסינון לפונקציה פרטית נפרדת. הדבר משפר את קריאות הקוד ומאפשר לבצע בדיקות יחידה (Unit Tests) על לוגיקת המרכיבים בקורס.
- **ניהול מחוזות (Coding Standards):** מחרוזות כגון "Hello" והודעות שהגיאה כתובות כתקסט חופשי (Hardcoded) בתוך הקוד. יש להעתיקן לקובץ res/values/strings.xml כדי לאפשר תמייה בריבוי שפות (לוקלייזציה) 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) לקובידינטות.

הדווחות שכתבנו על קוד של קבוצה אחרת:

דווח 1:

תז: 325445997

הקבצים שנבדקו: ReportItemViewModel, ReportFoundItemActivity
סוג הקוד - MVVM (UI + ViewModel)

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

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

נקודות חיוביות בקוד:

- שימוש ברור ועקבי בViewModel לצורך בידוד לוגיקה מה-UI.
- שמות משתנים ופונקציות קרייניטים וברורים.
- העורות ותייעוד איכוטי ומובן שמסביר את מטרת הפונקציות.
- אינטראקציה נכונה בין השכבות (Repository -> ViewModel -> UI)

הערות מרכזיות:

1. הפרדת אחריות - במהלך קריאת הקוד ניתן לראות שהחלק המלוגיקה שנמצאת בע-Activity אינה קשורה לשירות לאו, כמו המרת תמונה Base64 וחישוב קואורדינטות מתוך כתובת. פעולה אלו שייכות יותר לוגיקה עסקית, והימצאות בע-Activity יוצרת עומס יתר.

הצעה לשיפור - מומלץ להעביר פונקציות כמו `encodeImageToBase64` ו- `getLatLongFromAddress` ב-ViewModel למחיקות עצר ייעודית. כך הע-Activity תשאיר אחריה בעיקר על קלט ותצוגה, והקוד יהיה ברור וקל יותר לתחזקה.

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

הצעה לשיפור - כדי לשפר את הקריאות ולצמצם כפליות, ניתן לאחד את הלוגיקה המשותפת לפונקציה כללית אחת, או להשתמש במבנה משותף לייצור האובייקטים. שנית, זה יקצר את הקוד ויקל על שינויים עתידיים.

3. טיפול בשגיאות - במקרים שבהם מתרחש כשל בתהיליך (ולידציה, חישוב מיקום או שמירה 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 ה成败, להוסיף לוגים רלוונטיים, ולהציג למשתמש הודעה מתאימה במקרה של שגיאה. כך ניתן לשפר גם את חווית השימוש וגם את תהליך הדיבוג.

דוגמא לשיפור בעקבות ה-Review:

לפני השיפור, ה- Activity כללה לוגיקה של המרת תמונה וחישוב קואורדינטות. בהצעה לשיפור, לוגיקה זו מועברת ל- ViewModel, והע-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 והתחזקה של המערכת.

LoginActivity

א. בזרימת ההתחברות עם Google SignInAccount קיימת אפשרות ש'חזור null' במצבים מסוימים. במצב זהה קריאה ישירה ל-firebaseAuthWithGoogle עלולה לגרום לךיטה.

ב. יש כמה פונקציות כמו `performFirebaseLogin`, `attemptLogin`, `attemptSignIn` על קריית הקוד, אפשר להוסיף אפילו משפט אחד שמתאר את המטרה של הפונקציה.

ג. `LoginActivity` מרגע עמו מדי ומבצעת פעולות שלא קשורות לתצוגה, כמו תקשורת ישירה מול ה-`Firebase` API של העבר את כל הלוגיקה של `ViewModel` ל-`Activity`. כדי-

דוגמה לשיפור:

להוסיף בדיקה שהמשתנה `GoogleSignInAccount` לא null.

:3 דוח

תז: 322516840

Code Review Summary Document**Module:**

ReportLostItemActivity.java

Purpose:

מסך דיווח על פריט אבוד, הכוללת קליטת נתונים מהמשתמש, צילום תמונה, המרת תמונה ל-Base64 והמרת כתובות 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 כתובות Repository). לפ"י עקרונות MVVM, מומלץ להעביר את הלוגיקה זו ל-`ViewModel` (Lat/Lng).

ה-`Activity` אמרור להתמקד בקלט/תצוגה בלבד (Layer UI), בעוד שכבת ה-`ViewModel` אחראית על עיבוד הנתונים.

מומלץ לשיפור מודולריות, קלות תחזקה, בדיקות (Unit Tests) פשוטות יותר והקטנת עומס על שכבת ה-UI.

2. ניהול משאבים ויציבות

בפונקציה (`encodeImageToBase64`) נפתח `InputStream` אך איןנו נסגר בצורה מפורשת. זה מהוות דליית משאבי (Resource Leak) ועלול להוביל לביעות יציבות לאורך זמן.

בנוסף, פעולות כבדות כמו Base64-Bitmap processing או Base64-עלילות לגרום לעומס זיכרון ולירידה בביטויים.

המלצתה: שימוש ב `try-with-resources` (Background Thread) והעברת העבודה לרקע. (Background Thread)

3. טיפול בשגיאות וחווית משתמש

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

המלצתה:

- ולידציה לפני שליחה (עם הודעות שגיאה ספציפיות).

- הצגת Loading Indicator/Spinner בזמן עיבוד ושליחה.

- הרצת/Geocoder עיבוד תמורה ברקע.

השפעה: פחתת "תקינות", פחתת דיווחים כפולים, וחווית משתמש מkcטועית יותר.

Example of Improvement:

המעבר לוגיקת Base64 + Geocoding + Repository (או ViewModel-try-with-resources), ונוסף שימוש ב-Activity-submitForm-לפניה: כל העבודה התרחש ב-Activity, כולל גיאוקודינג וקידוד Base64. לאחר השינוי: רק אוסף נתונים גלמיים ומוריך ל-ViewModel, שמבצע את העבודה הכבודה ברקע. כתוצאה לכך הקוד נהייה קרייאטיבי, ניתן לתזוזקה ובניי בצורה נכונה מבחינה שכבות הארכיטקטורה.

Code Review Summary Document

Module:

ReportItemViewModel.java

Purpose:

viewModel שאחראי על הלוגיקה העוסקת בבדיקה פריטים אבודים/נמצאים: ולידציה לשדות חובה, בניית מודלים (LostItem / FoundItem) והאצלת פעולה העלאה ל-Repository (Firestore/Storage callback).

Key Comments:

מודולריות והפרחת כפליות

יש כפליות בין reportLostItem ל-reportFoundItem -

שתי הfonkcioot מבצעות את אותו שלבים כמעט בדילוק (ולידציה - יצרת מודל- שדות set-קריאה repository).

מומלץ לחלק את הלוגיקה המשותפת לפונקציה/Builder משותף, או לחלופין להשתמש בפולימורפיזם (Factory/Mapper) למושך באמצעות ממתק לשתי LostItem ו-FoundItem , או תבנית עיצוב מסוג (

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

וילידציה ובדיקות

הוילידציה (Valid) בודקת רק אם הערכים הם صحيحו ריקים, וכן חסרים מקרים נפוצים שלולים callback לשגיאות: למשל מחוזות שמכילים רק רווחים עדין "חובב כ'לא ריקות", יתכן שה- callback.accept(...) יgive כ- nullvoid קריאה ל- (...) callback.setImageBase64imageUri גורם ל- NPE(תליי בימוש של המודל), ובנוספ עריכ Ing/latitudelongitude מועברים אך במקורה שה- Geocoder נ>null הפועלות שלותה 0.0/0.0 נראה "תיקין" למרות שמדובר במקומות לא אמיתיים ולפיכך לא ניתן בורורה האם לאפשר ערך זה או להציג כישלון/ازהרה; לכן מומלץ להשתמש ב- trim() בדיקות הירוקות, להגן מפני מצב שבו callbacknullvoid האות callbacknullboolean (למשל nullable boolean hasCoordinates או שימוש בערכים), מה שיקטין קרטיסות ומנע שמירה של דיווחים שגויים עם מקום לא אמיתי.

ניהול תלוויות ותשתיות

במחלקה הנוכחית ה- ViewModel יוצר את ה- Repository בתוך עצמו באמצעות private final ItemRepository repository = new ItemRepository();, (dependency injection) מושם שלא ניתן להחליף את ה- Mocking (模拟) ופוגע בעיקרונו של Repository במשימוש חלופי לצורך טסטים או הרחבה; בנוסף, השימוש ב- Consumer<Boolean> הוא מגנון callback של Java 8 אשר בפיתוח Android מודרני נהוג לעבוד עם callbacks מותאמים או עם מגנונים כמו LiveData / StateFlow שמשופקים זרימה מסודרת של מצב למסך, וכן מומלץ להזrik את ה- Repository דרך ה- ViewModel Factory או לאפשר לאפשר Constructor/constructor/package-private setter<Result> State<State> אשר יוכל להציג טענה/שגיאה/הצלחה בצורה עקבית ווטנדרטית.

Example of Improvement:

השינוי שבוצע כולל חיזוק מגנון הוילידציה והגנה מפני מצב שבו callbacknullvoid תיקון מקירה של שדות שמכילים רווחים בלבד. לפני השיפור, הפונקציה ()isValid/empty, רק callback.setImageBase64imageUri מחרוזות כמו " עברו קקלט תקין, ובנוספ אם ה- callback.accept(...) יgive כ- nullvoid קריאה ל- (...) callback.setImageBase64imageUri גורמת לקריישה. לאחר השיפור, מבצעים ()trim callbacknullvoid שדה טקסט לפני בדיקות הוילידציה ומוציאים בדיקת guard שמבודדת שה- callbacknullvoid לא יכול להציג שימוש בו. כתוצאה לכך מתאפשר קוד יציב יותר, עם פחות קרייסות, ולידציה נכונה שמונעת שמירת דיווחים "ריקים" בפועל.