Projet 5 : Délivrez une application mobile en production

Cleanup

Contexte :

Bonjour,

Je m'appelle Christine et je suis la fondatrice de Cleanup, une agence de nettoyage de locaux commerciaux.

Nous avons fait développer une application par un de nos prestataires. Elle s'appelle Todoc et permet à chacun de nos collaborateurs de gérer une liste de tâches quotidiennes.

Le prestataire a rompu son contrat avant de terminer sa mission. Il n'a pas effectué la persistance des données. Vous voyez l'urgence de la situation !

Contexte :

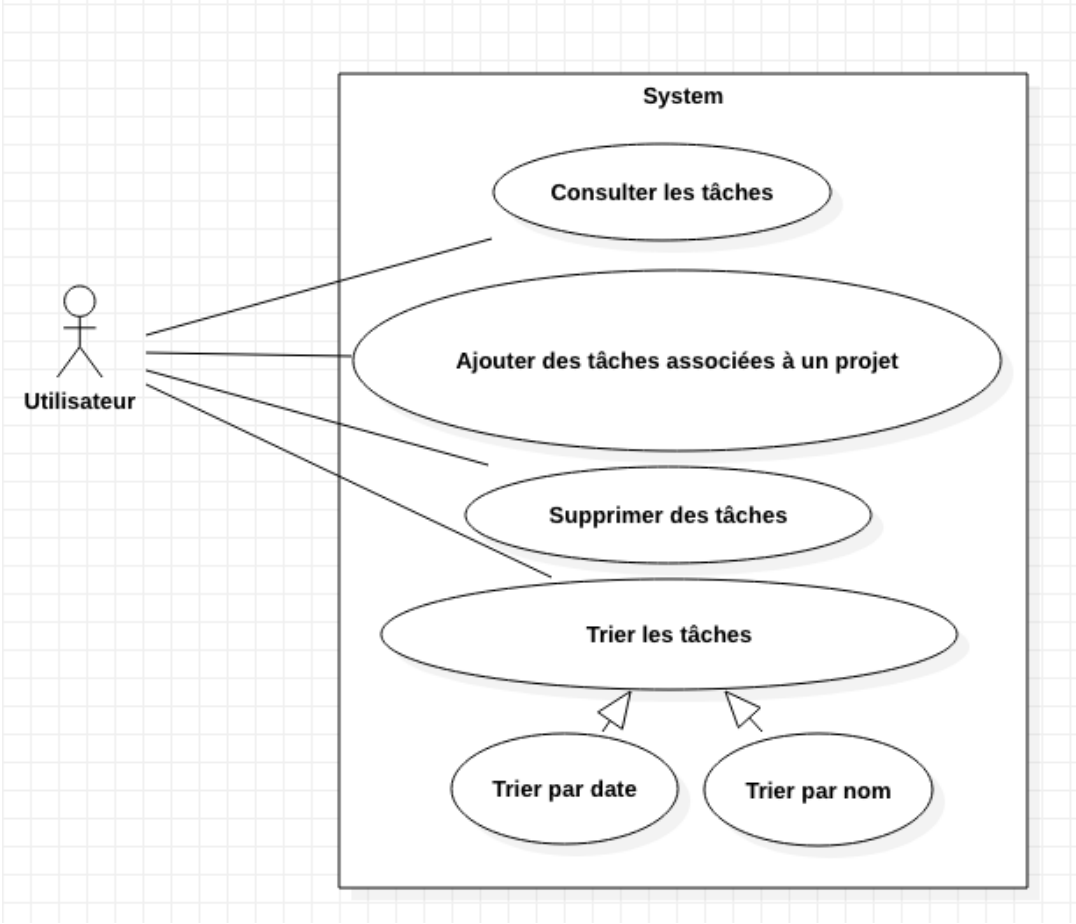
Mission à réaliser

- Gérer la persistance des données de l'application.

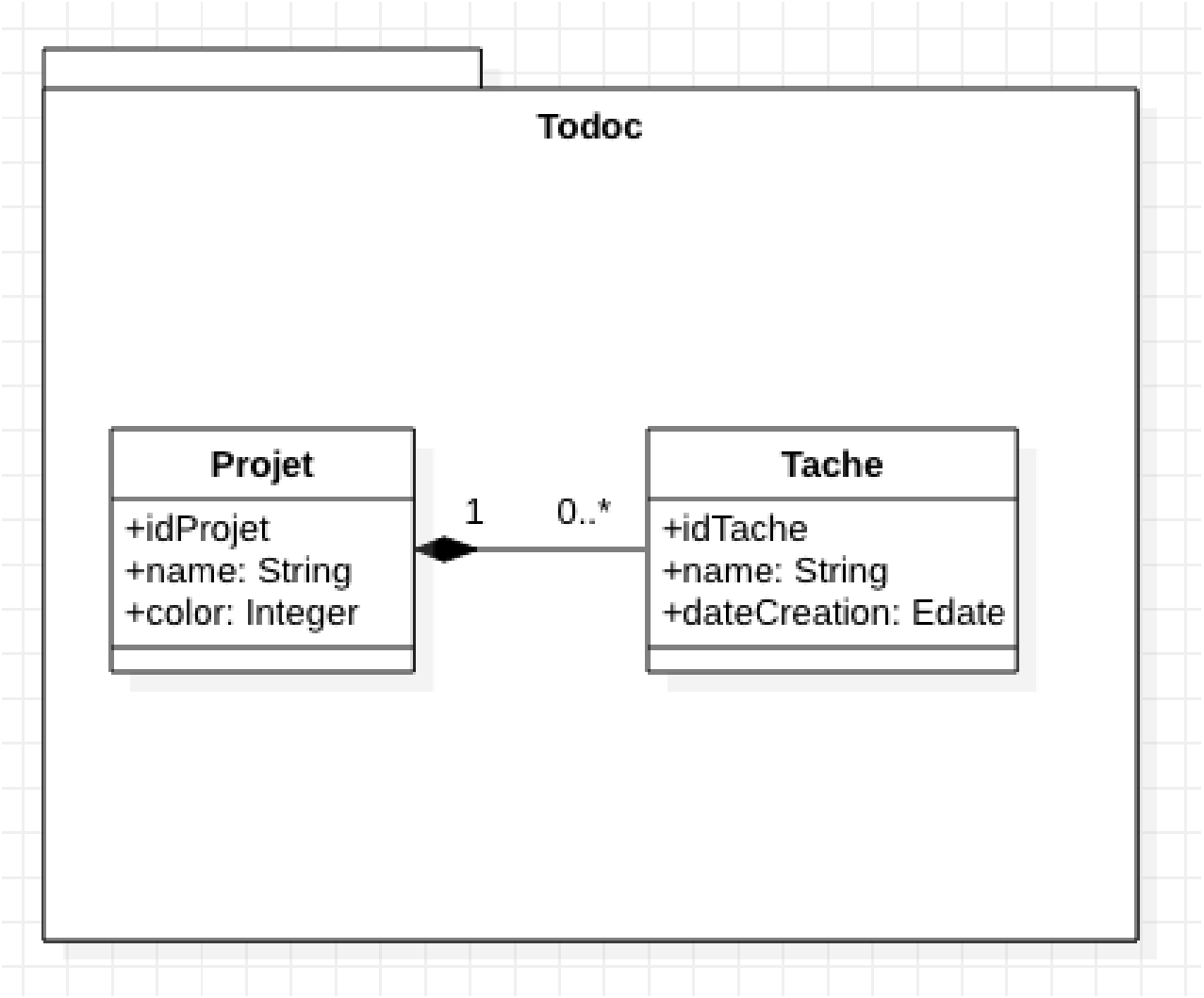
Contraintes

- Implémentation d'une base de données SQLite. Objectif : sauvegarde de la liste des tâches facilitée et structurée.
- Réalisation d'un modèle relationnel.
- Réalisation d'un diagramme de classes. Objectif : mettre en avant les différentes classes utilisées dans l'application et les relations entre elles.
- Réalisation d'un diagramme d'utilisation. Objectif : mieux visualiser le comportement fonctionnel de l'application.
- Déploiement de l'application sur le Play Store en mode "bêta", avec création d'une version "release" et "signée", en générant l'APK correspondant.
- Code l'application obfusqué en version "release", afin de protéger encore un peu plus le code source de ce projet.
- Mise à jour des tests, afin d'intégrer SQLite à l'application.

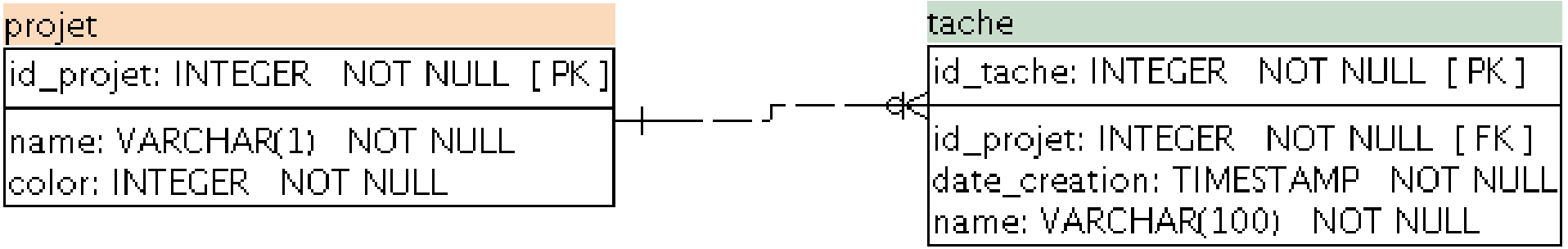
Réalisation d'un diagramme de cas d'utilisation :



Réalisation d'un diagramme de classes.



Réalisation d'un modèle relationnel.



(Model) Project :

```java
@Entity
public class Project {
    /**
     * The unique identifier of the project
     */
    @PrimaryKey
    private final long id;

    /**
     * The name of the project
     */
    @NonNull
    private final String name;

    /**
     * The hex (ARGB) code of the color associated to the project
     */
    @ColorInt
    private final int color;

    /**
     * Instantiates a new Project.
     *
     * @param id    the unique identifier of the project to set
     * @param name  the name of the project to set
     * @param color the hex (ARGB) code of the color associated to the project to
     */
    public Project(long id, @NonNull String name, @ColorInt int color) {
        this.id = id;
        this.name = name;
        this.color = color;
    }

    /**
     * Returns all the projects of the application.
     *
     * @return all the projects of the application
     */
    @NonNull
    public static Project[] getAllProjects() {
        return new Project[]{
                new Project( id: 1L,  name: "Projet Tartampion",  color: 0xFFEADAD1),
                new Project( id: 2L,  name: "Projet Lucidia",  color: 0xFFB4CDBA),
                new Project( id: 3L,  name: "Projet Circus",  color: 0xFFA3CED2),
        };
    }
```

```java
/**
 * Returns the project with the given unique identifier, or null if no project with that
 * identifier can be found.
 *
 * @param id the unique identifier of the project to return
 * @return the project with the given unique identifier, or null if it has not been foun
 */
@Nullable
public static Project getProjectById(long id) {
    for (Project project : getAllProjects()) {
        if (project.id == id)
            return project;
    }
    return null;
}

/**
 * Returns the unique identifier of the project.
 *
 * @return the unique identifier of the project
 */
public long getId() { return id; }

/**
 * Returns the name of the project.
 *
 * @return the name of the project
 */
@NonNull
public String getName() { return name; }

/**
 * Returns the hex (ARGB) code of the color associated to the project.
 *
 * @return the hex (ARGB) code of the color associated to the project
 */
@ColorInt
public int getColor() { return color; }

@Override
@NonNull
public String toString() { return getName(); }
```

(Model) Task :

```java
@Entity(foreignKeys = @ForeignKey(entity = Project.class,
        parentColumns = "id",
        childColumns = "projectId"))
public class Task {
    /**
     * The unique identifier of the task
     */
    @PrimaryKey(autoGenerate = true)
    private long id;

    /**
     * The unique identifier of the project associated to the task
     */
    private long projectId;

    /**
     * The name of the task
     */
    // Suppress warning because setName is called in constructor
    @SuppressWarnings("NullableProblems")
    @NonNull
    private String name;

    /**
     * The timestamp when the task has been created
     */
    private long creationTimestamp;

    /**
     * Instantiates a new Task.
     *
     * @param id                the unique identifier of the task to set
     * @param projectId         the unique identifier of the project associated to the ta
     * @param name              the name of the task to set
     * @param creationTimestamp the timestamp when the task has been created to set
     */
    public Task(long id, long projectId, @NonNull String name, long creationTimestamp) {
        this.setId(id);
        this.setProjectId(projectId);
        this.setName(name);
        this.setCreationTimestamp(creationTimestamp);
    }
```

```java
public long getId() { return id; }
public long getProjectId() { return projectId; }
public long getCreationTimestamp() { return creationTimestamp; }

/**
 * Sets the unique identifier of the task.
 *
 * @param id the unique idenifier of the task to set
 */
private void setId(long id) { this.id = id; }

/**
 * Sets the unique identifier of the project associated to the task.
 *
 * @param projectId the unique identifier of the project associated to the task to set
 */
private void setProjectId(long projectId) { this.projectId = projectId; }

/**
 * Returns the project associated to the task.
 *
 * @return the project associated to the task
 */
@Nullable
public Project getProject() { return Project.getProjectById(projectId); }

/**
 * Returns the name of the task.
 *
 * @return the name of the task
 */
@NonNull
public String getName() { return name; }

/**
 * Sets the name of the task.
 *
 * @param name the name of the task to set
 */
private void setName(@NonNull String name) { this.name = name; }
```

(Model) Task (suite):

```
private void setCreationTimestamp(long creationTimestamp) {
    this.creationTimestamp = creationTimestamp;
}

/**
 * Comparator to sort task from A to Z
 */
public static class TaskAZComparator implements Comparator<Task> {
    @Override
    public int compare(Task left, Task right) { return left.name.compareTo(right.name); }
}

/**
 * Comparator to sort task from Z to A
 */
public static class TaskZAComparator implements Comparator<Task> {
    @Override
    public int compare(Task left, Task right) { return right.name.compareTo(left.name); }
}

/**
 * Comparator to sort task from last created to first created
 */
public static class TaskRecentComparator implements Comparator<Task> {
    @Override
    public int compare(Task left, Task right) {
        return (int) (right.creationTimestamp - left.creationTimestamp);
    }
}

/**
 * Comparator to sort task from first created to last created
 */
public static class TaskOldComparator implements Comparator<Task> {
    @Override
    public int compare(Task left, Task right) {
        return (int) (left.creationTimestamp - right.creationTimestamp);
    }
}
```

ProjectDao et TaskDao :

```java
@Dao
public interface ProjectDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void createProject(Project project);

    @Query("SELECT * FROM Project ")
    LiveData<List<Project>>  getProjects();
}
```

```java
@Dao
public interface TaskDao {

    @Query("SELECT * FROM Task ")
    LiveData<List<Task>> getTasks();

    @Insert
    long insertTask(Task item);


    @Query("DELETE FROM Task WHERE id = :taskId")
    int deleteTask(long taskId);
}
```

SaveMyData :

```java
@Database(entities = {Task.class, Project.class},
        version = 1, exportSchema = false)
public abstract class SaveMyData extends RoomDatabase {

    // --- SINGLETON ---
    private static volatile SaveMyData INSTANCE;

    // --- DAO ---
    public abstract TaskDao taskDao();
    public abstract ProjectDao projectDao();

    // --- INSTANCE ---
    public static SaveMyData getInstance(Context context) {
        if (INSTANCE == null) {
            synchronized (SaveMyData.class) {
                if (INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(context.
                                    getApplicationContext(),
                            SaveMyData.class, name: "Database.db")
                            .addCallback(prepopulateDatabase())
                            .build();
                }
            }
        }
        return INSTANCE;
    }
```

```java
private static Callback prepopulateDatabase(){
    return onCreate(db) → {

                super.onCreate(db);

            ContentValues contentValues = new ContentValues();
            contentValues.put("id", 1L);
            contentValues.put("name", "Projet Tartampion");
            contentValues.put("color", 0xFFEADAD1);

            db.insert( table: "Project", OnConflictStrategy.IGNORE, contentValues);

            ContentValues contentValues2 = new ContentValues();
            contentValues2.put("id", 2L);
            contentValues2.put("name", "Projet Lucidia");
            contentValues2.put("color", 0xFFB4CDBA);

            db.insert( table: "Project", OnConflictStrategy.IGNORE, contentValues2);

            ContentValues contentValues3 = new ContentValues();
            contentValues3.put("id", 3L);
            contentValues3.put("name", "Projet Circus");
            contentValues3.put("color", 0xFFA3CED2);

            db.insert( table: "Project", OnConflictStrategy.IGNORE, contentValues3);
    };
}
```

ProjectDataRepository et TaskDataRepository :

```java
public class ProjectDataRepository {


    private final ProjectDao projectDao;

    public ProjectDataRepository(ProjectDao projectDao) { this.projectDao = projectDao; }

    // --- GET PROJECT ---
    public LiveData<List<Project>>  getProjects() { return this.projectDao.getProjects(); }


}
```

```java
public class TaskDataRepository {

    private final TaskDao taskDao;

    public TaskDataRepository(TaskDao taskDao) { this.taskDao = taskDao; }

    // --- GET ---

    public LiveData<List<Task>> getTasks(){ return this.taskDao.getTasks(); }

    // --- CREATE ---

    public void createTask(Task task){ taskDao.insertTask(task); }

    // --- DELETE ---
    public void deleteTask(long taskId){ taskDao.deleteTask(taskId); }


}
```

TaskViewModel :

```java
public class TaskViewModel extends ViewModel {

    // REPOSITORIES
    private final TaskDataRepository taskDataSource;
    private final ProjectDataRepository projectDataSource;
    private final Executor executor;

    // DATA
    @Nullable
    private LiveData<List<Project>> currentProject;

    public TaskViewModel(TaskDataRepository taskDataSource, ProjectDataRepository projectDataSource, Executor executor) {
        this.taskDataSource = taskDataSource;
        this.projectDataSource = projectDataSource;
        this.executor = executor;
    }

    public void init() {
        if (this.currentProject != null) {
            return;
        }
        currentProject = projectDataSource.getProjects();
    }

    // -------------
    // FOR PROJECT
    // -------------

    public LiveData<List<Project>> getProjects() { return this.currentProject;  }

    // -------------
    // FOR TASK
    // -------------

    public LiveData<List<Task>> getTasks() { return taskDataSource.getTasks() ; }

    public void createTask(Task task) {
        executor.execute(() -> {
            taskDataSource.createTask(task);
        });
    }

    public void deleteTask(long taskId) {
        executor.execute(() -> {
            taskDataSource.deleteTask(taskId);
        });
```

ViewModelFactory :

```java
public class ViewModelFactory implements ViewModelProvider.Factory {

    private final TaskDataRepository taskDataSource;
    private final ProjectDataRepository projectDataSource;
    private final Executor executor;

    public ViewModelFactory(TaskDataRepository taskDataSource, ProjectDataRepository projectDataSource, Executor executor) {
        this.taskDataSource = taskDataSource;
        this.projectDataSource = projectDataSource;
        this.executor = executor;
    }

    @Override
    public <T extends ViewModel> T create(Class<T> modelClass) {
        if (modelClass.isAssignableFrom(TaskViewModel.class)) {
            return (T) new TaskViewModel(taskDataSource, projectDataSource, executor);
        }
        throw new IllegalArgumentException("Unknown ViewModel class");
    }
}
```

Injection :

```java
public class Injection {

    public static TaskDataRepository provideItemDataSource(Context context) {
        SaveMyData database = SaveMyData.getInstance(context);
        return new TaskDataRepository(database.taskDao());
    }

    public static ProjectDataRepository provideUserDataSource(Context context) {
        SaveMyData database = SaveMyData.getInstance(context);
        return new ProjectDataRepository(database.projectDao());
    }

    public static Executor provideExecutor(){ return Executors.newSingleThreadExecutor(); }

    public static ViewModelFactory provideViewModelFactory(Context context) {
        TaskDataRepository dataSourceItem = provideItemDataSource(context);
        ProjectDataRepository dataSourceUser = provideUserDataSource(context);
        Executor executor = provideExecutor();
        return new ViewModelFactory(dataSourceItem, dataSourceUser, executor);
    }
}
```

TaskAdapter (pas de changement) :

```java
public class TasksAdapter extends RecyclerView.Adapter<TasksAdapter.TaskViewHolder> {
    /**
     * The list of tasks the adapter deals with
     */
    @NonNull
    private List<Task> tasks = new ArrayList<~>() ;

    /**
     * The listener for when a task needs to be deleted
     */
    @NonNull
    private final DeleteTaskListener deleteTaskListener;

    /**
     * Instantiates a new TasksAdapter.
     *
     */
    TasksAdapter( @NonNull final DeleteTaskListener deleteTaskListener) {
       // this.tasks = tasks;
        this.deleteTaskListener = deleteTaskListener;
    }

    /**
     * Updates the list of tasks the adapter deals with.
     *
     * @param tasks the list of tasks the adapter deals with to set
     */
    void updateTasks(@NonNull final List<Task> tasks) {
        this.tasks = tasks;
        notifyDataSetChanged();
    }

    @NonNull
    @Override
    public TaskViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int viewType) {
        View view = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_task, viewGroup, false);
        return new TaskViewHolder(view, deleteTaskListener);
    }

    @Override
    public void onBindViewHolder(@NonNull TaskViewHolder taskViewHolder, int position) {
        taskViewHolder.bind(tasks.get(position));
    }

    @Override
    public int getItemCount() { return tasks.size(); }
```

```java
/**
 * Listener for deleting tasks
 */
public interface DeleteTaskListener {
    /**
     * Called when a task needs to be deleted.
     *
     * @param task the task that needs to be deleted
     */
    void onDeleteTask(Task task);
}

/**
 * <p>ViewHolder for task items in the tasks list</p>
 *
 * @author Gaëtan HERFRAY
 */
class TaskViewHolder extends RecyclerView.ViewHolder {
    /**
     * The circle icon showing the color of the project
     */
    private final AppCompatImageView imgProject;

    /**
     * The TextView displaying the name of the task
     */
    private final TextView lblTaskName;

    /**
     * The TextView displaying the name of the project
     */
    private final TextView lblProjectName;

    /**
     * The delete icon
     */
    private final AppCompatImageView imgDelete;

    /**
     * The listener for when a task needs to be deleted
     */
    private final DeleteTaskListener deleteTaskListener;
```

TaskAdapter (pas de changement) :

```
     * @param itemView the view of the task item
     * @param deleteTaskListener the listener for when a task needs to be deleted to set
     */
    TaskViewHolder(@NonNull View itemView, @NonNull DeleteTaskListener deleteTaskListener) {
        super(itemView);

        this.deleteTaskListener = deleteTaskListener;

        imgProject = itemView.findViewById(R.id.img_project);
        lblTaskName = itemView.findViewById(R.id.lbl_task_name);
        lblProjectName = itemView.findViewById(R.id.lbl_project_name);
        imgDelete = itemView.findViewById(R.id.img_delete);

        imgDelete.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                final Object tag = view.getTag();
                if (tag instanceof Task) {
                    TaskViewHolder.this.deleteTaskListener.onDeleteTask((Task) tag);
                }
            }
        });
    }

    /**
     * Binds a task to the item view.
     *
     * @param task the task to bind in the item view
     */
    void bind(Task task) {
        lblTaskName.setText(task.getName());
        imgDelete.setTag(task);

        final Project taskProject = task.getProject();
        if (taskProject != null) {
            imgProject.setSupportImageTintList(ColorStateList.valueOf(taskProject.getColor()));
            lblProjectName.setText(taskProject.getName());
        } else {
            imgProject.setVisibility(View.INVISIBLE);
            lblProjectName.setText("");
        }

    }
}
```

MainActivity :

```
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);


    listTasks = findViewById(R.id.list_tasks);

    lblNoTasks = findViewById(R.id.lbl_no_task);
    lblNoTasks.setVisibility(View.INVISIBLE);


    listTasks.setLayoutManager(new LinearLayoutManager( context: this,
            LinearLayoutManager.VERTICAL,  reverseLayout: false));
    listTasks.setAdapter(adapter);



    this.configureViewModel();


    findViewById(R.id.fab_add_task).setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) { showAddTaskDialog(); }
    });

    sharedPref = getBaseContext().getSharedPreferences( name: "TheFileName",
                    Context.MODE_PRIVATE);

    String sortMethodString = sharedPref.getString(FAVORITE_SORTING_METHOD,
             defValue: "OLD_FIRST");


    switch (sortMethodString) {
        case "ALPHABETICAL":
            sortMethod = SortMethod.ALPHABETICAL;
            break;
        case "ALPHABETICAL_INVERTED":
            sortMethod = SortMethod.ALPHABETICAL_INVERTED;
            break;
        case "OLD_FIRST":
            sortMethod = SortMethod.OLD_FIRST;
            break;
        case "RECENT_FIRST":
            sortMethod = SortMethod.RECENT_FIRST;
            break;
    }
    this.getTasks();
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.actions, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {



    SharedPreferences.Editor editor = sharedPref.edit();

    int id = item.getItemId();

    if (id == R.id.filter_alphabetical) {
        sortMethod = SortMethod.ALPHABETICAL;
        editor.putString(FAVORITE_SORTING_METHOD, "ALPHABETICAL");
    } else if (id == R.id.filter_alphabetical_inverted) {
        sortMethod = SortMethod.ALPHABETICAL_INVERTED;
        editor.putString(FAVORITE_SORTING_METHOD, "ALPHABETICAL_INVERTED");
    } else if (id == R.id.filter_oldest_first) {
        sortMethod = SortMethod.OLD_FIRST;
        editor.putString(FAVORITE_SORTING_METHOD, "OLD_FIRST");
    } else if (id == R.id.filter_recent_first) {
        sortMethod = SortMethod.RECENT_FIRST;
        editor.putString(FAVORITE_SORTING_METHOD, "RECENT_FIRST");

    }

    editor.apply();


    updateTasks();

    return super.onOptionsItemSelected(item);
}

@Override
public void onDeleteTask(Task task) {
    this.taskViewModel.deleteTask(task.getId());
    tasks -= 1;

    updateTasks();

}
```

MainActivity :

```
private void onPositiveButtonClick(DialogInterface dialogInterface) {
    // If dialog is open
    if (dialogEditText != null && dialogSpinner != null) {
        // Get the name of the task
        String taskName = dialogEditText.getText().toString();

        // Get the selected project to be associated to the task
        Project taskProject = null;
        if (dialogSpinner.getSelectedItem() instanceof Project) {
            taskProject = (Project) dialogSpinner.getSelectedItem();
        }

        // If a name has not been set
        if (taskName.trim().isEmpty()) {
            dialogEditText.setError(getString(R.string.empty_task_name));
        }
        // If both project and name of the task have been set
        else if (taskProject != null) {

            Task task = new Task( id: 0,
                    taskProject.getId(),
                    taskName,
                    new Date().getTime()
            );

            addTask(task);


            dialogInterface.dismiss();
        }
        // If name has been set, but project has not been set (this should never occur)
        else{
            dialogInterface.dismiss();
        }
    }
    // If dialog is aloready closed
    else {
        dialogInterface.dismiss();
    }
}
```

```
private void showAddTaskDialog() {
    final AlertDialog dialog = getAddTaskDialog();

    dialog.show();

    dialogEditText = dialog.findViewById(R.id.txt_task_name);
    dialogSpinner = dialog.findViewById(R.id.project_spinner);

    populateDialogSpinner();
}

private void addTask(@NonNull Task task) {
    this.taskViewModel.createTask(task);
    tasks += 1;
    updateTasks();
}

private void updateTasks() {


    if (tasks == 0) {
        lblNoTasks.setVisibility(View.VISIBLE);
        listTasks.setVisibility(View.GONE);
    } else {
        lblNoTasks.setVisibility(View.GONE);
        listTasks.setVisibility(View.VISIBLE);

        switch (sortMethod) {
            case ALPHABETICAL:
                Collections.sort(mlistTask, new Task.TaskAZComparator());
                break;
            case ALPHABETICAL_INVERTED:
                Collections.sort(mlistTask, new Task.TaskZAComparator());
                break;
            case RECENT_FIRST:
                Collections.sort(mlistTask, new Task.TaskRecentComparator());
                break;
            case OLD_FIRST:
                Collections.sort(mlistTask, new Task.TaskOldComparator());
                break;
        }
        adapter.updateTasks(mlistTask);
    }
}
```

MainActivity :

```
private AlertDialog getAddTaskDialog() {
    final AlertDialog.Builder alertBuilder = new AlertDialog.Builder( context: this, R.style.Dialog);

    alertBuilder.setTitle(R.string.add_task);
    alertBuilder.setView(R.layout.dialog_add_task);
    alertBuilder.setPositiveButton(R.string.add, null);
    alertBuilder.setOnDismissListener(new DialogInterface.OnDismissListener() {
        @Override
        public void onDismiss(DialogInterface dialogInterface) {
            dialogEditText = null;
            dialogSpinner = null;
            dialog = null;
        }
    });

    dialog = alertBuilder.create();

    // This instead of listener to positive button in order to avoid automatic dismiss
    dialog.setOnShowListener(new DialogInterface.OnShowListener() {

        @Override
        public void onShow(DialogInterface dialogInterface) {

            Button button = dialog.getButton(AlertDialog.BUTTON_POSITIVE);
            button.setOnClickListener(new View.OnClickListener() {

                @Override
                public void onClick(View view) { onPositiveButtonClick(dialog); }
            });
        }
    });

    return dialog;
}

/**
 * Sets the data of the Spinner with projects to associate to a new task
 */
private void populateDialogSpinner() {
    final ArrayAdapter<Project> adapterSpinner = new ArrayAdapter<>( context: this,
            android.R.layout.simple_spinner_item, allProjects);
    adapterSpinner.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    if (dialogSpinner != null) {
        dialogSpinner.setAdapter(adapterSpinner);
    }
}
```

```
/**
 * List of all possible sort methods for task
 */
private enum SortMethod {
    /**
     * Sort alphabetical by name
     */
    ALPHABETICAL,
    /**
     * Inverted sort alphabetical by name
     */
    ALPHABETICAL_INVERTED,
    /**
     * Lastly created first
     */
    RECENT_FIRST,
    /**
     * First created first
     */
    OLD_FIRST,
    /**
     * No sort
     */
    NONE
}

private void configureViewModel(){
    ViewModelFactory mViewModelFactory = Injection.
            provideViewModelFactory( context: this);
    this.taskViewModel = ViewModelProviders.
            of( activity: this, mViewModelFactory).get(TaskViewModel.class);
    this.taskViewModel.init();

}

// 3 - Get all tasks
private void getTasks(){
    this.taskViewModel.getTasks().observe( owner: this, this::updateTasksList);
}

private void updateTasksList(List<Task> listTask) {

    this.adapter.updateTasks(listTask);
    mlistTask = listTask;
    tasks = adapter.getItemCount();
    updateTasks();

}
```

TaskDaoTest :

```java
@RunWith(AndroidJUnit4.class)
public class TaskDaoTest {

    // FOR DATA
    private SaveMyData database;

    // DATA SET FOR TEST
    private static long PROJECT_ID = 1L;
    private static Project PROJECT_DEMO = new Project(PROJECT_ID, name: "Projet Tartampion",  color: 0xFFEADAD1);
    private static Task NEW_TASK_VAISELLE = new Task( id: 1,PROJECT_ID,  name: "faire la vaiselle",  creationTimestamp:
    private static Task NEW_TASK_SOL = new Task( id: 2,PROJECT_ID,  name: "laver le sol",  creationTimestamp: 13-12-2019
    private static Task NEW_TASK_ADMINISTRATIF = new Task( id: 3,PROJECT_ID,  name: "engager une nouvelle nettoyeuse"


    @Rule
    public InstantTaskExecutorRule instantTaskExecutorRule = new InstantTaskExecutorRule();

    @Before
    public void initDb() throws Exception {
        this.database = Room.inMemoryDatabaseBuilder(InstrumentationRegistry.getContext(),
                SaveMyData.class)
                .allowMainThreadQueries()
                .build();
    }

    @After
    public void closeDb() throws Exception {
        database.close();
    }


    @Test
    public void GetProjects() throws InterruptedException {
        this.database.projectDao().createProject(PROJECT_DEMO);
        List<Project> projects = LiveDataTestUtil.getValue(this.database.projectDao().getProjects());
        assertTrue( condition: projects.get(0).getName().equals(PROJECT_DEMO.getName()) && projects.get(0).getId() ==
    }


    @Test
    public void getTasksWhenNoTaskInserted() throws InterruptedException {
        // TEST
        List<Task> tasks = LiveDataTestUtil.getValue(this.database.taskDao().getTasks());
        assertTrue(tasks.isEmpty());
    }
```

TaskDaoTest :

```java
@Test
public void insertAndGetTask() throws InterruptedException {

    this.database.projectDao().createProject(PROJECT_DEMO);


    this.database.taskDao().insertTask(NEW_TASK_VAISELLE);
    this.database.taskDao().insertTask(NEW_TASK_SOL);
    this.database.taskDao().insertTask(NEW_TASK_ADMINISTRATIF);

    // TEST
    List<Task> tasks = LiveDataTestUtil.getValue(this.database.taskDao().getTasks());
    assertTrue( condition: tasks.size() == 3);
}

@Test
public void insertAndDeleteTask() throws InterruptedException {

    this.database.projectDao().createProject(PROJECT_DEMO);


    this.database.taskDao().insertTask(NEW_TASK_VAISELLE);
    Task taskAdded = LiveDataTestUtil.getValue(this.database.taskDao().getTasks()).get(0);
    this.database.taskDao().deleteTask(taskAdded.getId());

    //TEST
    List<Task> tasks = LiveDataTestUtil.getValue(this.database.taskDao().getTasks());
    assertTrue(tasks.isEmpty());
}
```

les rapports d'exécution (unitaire et instrumentalisé) des tests finaux qui réussissent :

MainActivityInstrume...: 2 total, 2 passed

com.cleanup.todoc.MainActivityInstrumentedTest

- addAndRemoveTask
- sortTasks

TaskDaoTest: 4 total, 4 passed

com.cleanup.todoc.TaskDaoTest

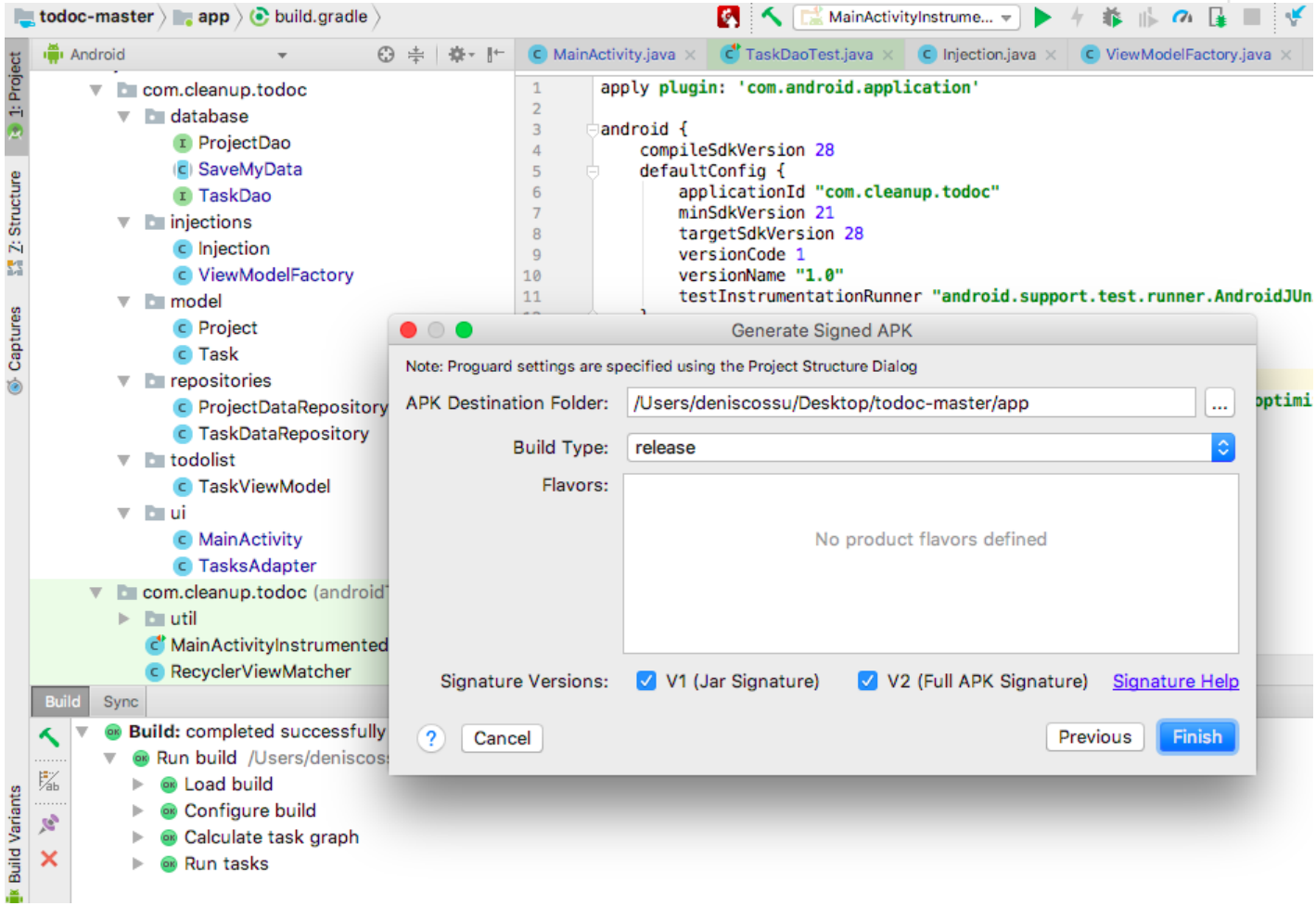
- GetProjects
- insertAndGetTask
- insertAndDeleteTask
- getTasksWhenNoTaskInserted

TaskUnitTest: 5 total, 5 passed

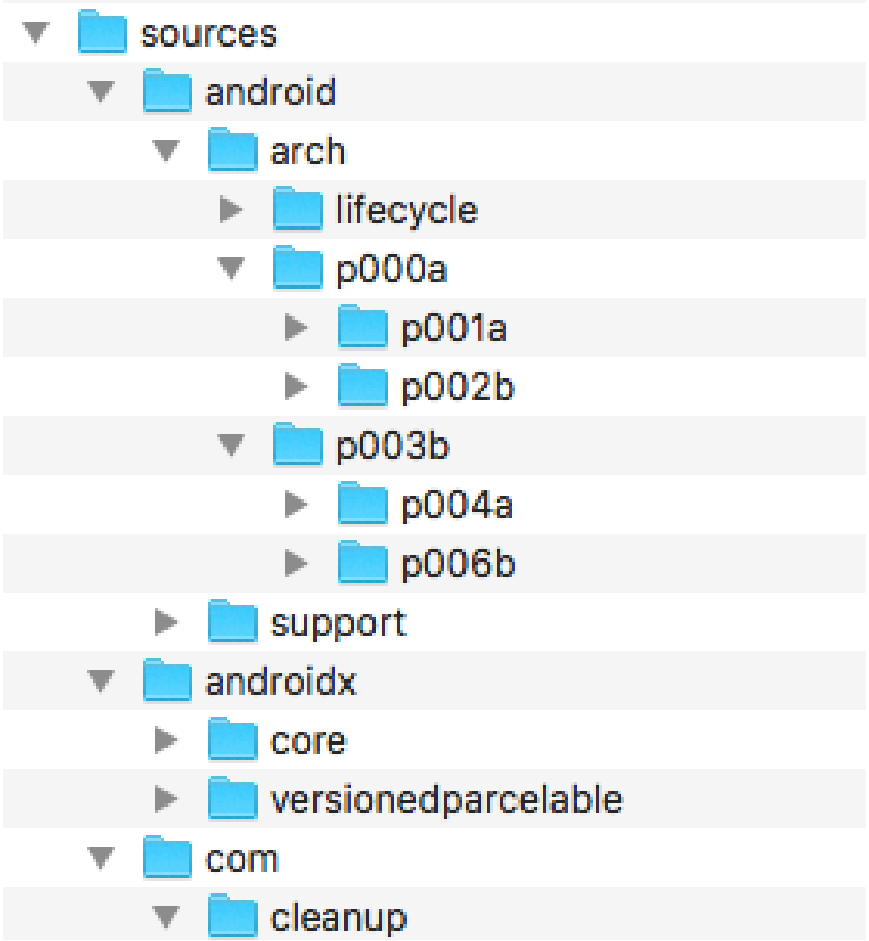
"/Applications/Android Studio.app/Contents/jre/jdk/Conter
-Didea.test.cyclic.buffer.size=1048576 -Didea.launcher.port=63
Studio.app/Contents/bin" -Dfile.encoding=UTF-8 -classpa

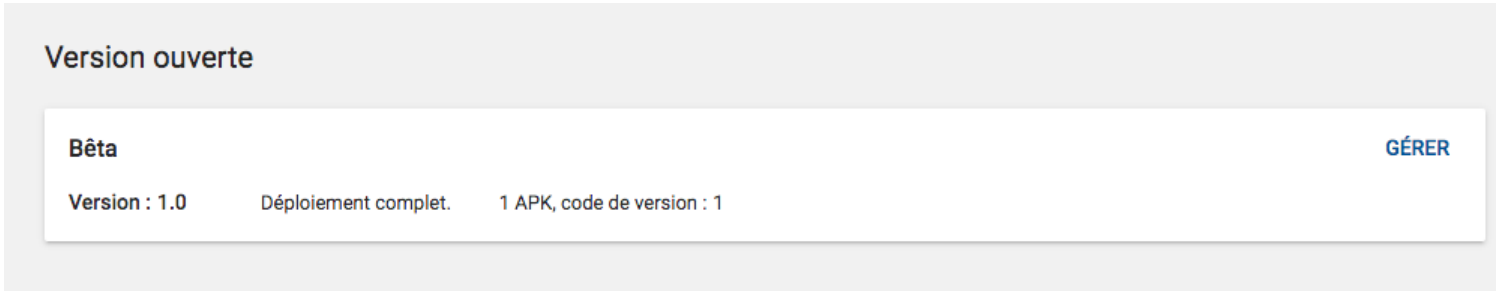
```
}
buildTypes {
    release {
        minifyEnabled true
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro
    }
}
```

Code de l'application obfusqué et génération de l'APK



On vérifie que le code est bien obfusqué





Application sur le PlayStore en mode « Bêta »