# QA Automation Exam: "TaskBoard"

**Exam** assignment for the [“QA Automation” Course @ SoftUni](https://softuni.bg/trainings/3767/qa-automation-may-2022).

Please submit your work as a single zip / rar / 7z archive holding the source code for each problem, without the binaries (exclude bin, obj, node\_modules and other unneeded folders).

## The "Task Board" System

“**Task Board**” is a simple information system for managing **tasks in a task board**. Each task consists of **title** + **description**. Tasks are organized in **boards**, which are displayed as columns (sections): Open, In Progress, Done. Users can **view** the task board with the tasks, **search** for tasks by keyword, **view** task details, **create** new tasks and **edit** existing tasks (and move existing tasks from one board to another).

You are given the RESTful **API** + **Web** client app + **Desktop** client app + **Android** mobile app client for the task board system. Your assignment is to write **API tests and UI automated tests** for the system.

You are given the following project assets:

* <https://github.com/nakov/TaskBoard> – source code of the TaskBoard Web client app and RESTful API
* <https://github.com/nakov/TaskBoard-DesktopClient> – TaskBoard desktop app for Windows
* <https://github.com/nakov/TaskBoard-AndroidClient> – TaskBoard mobile app for Android

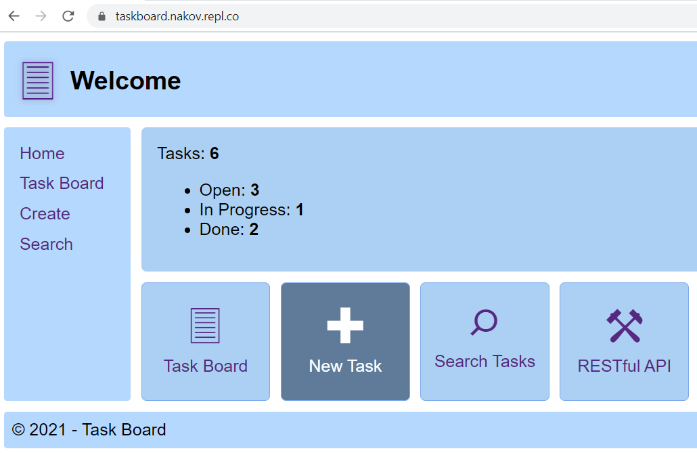
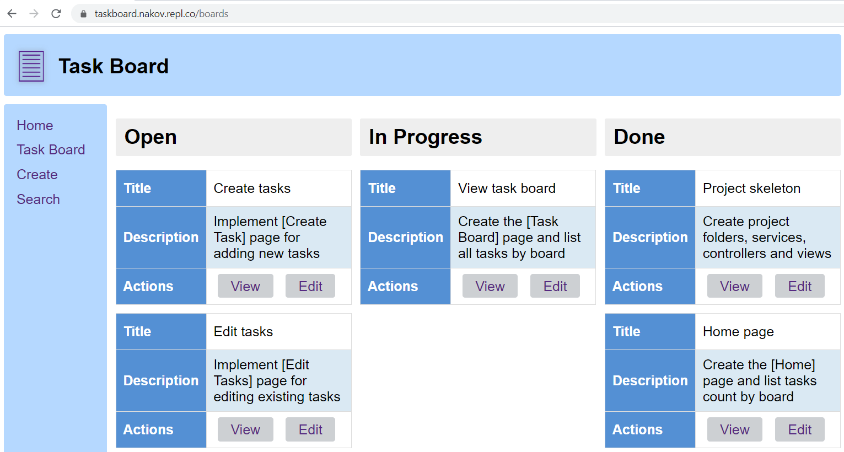
### Web App Functionality

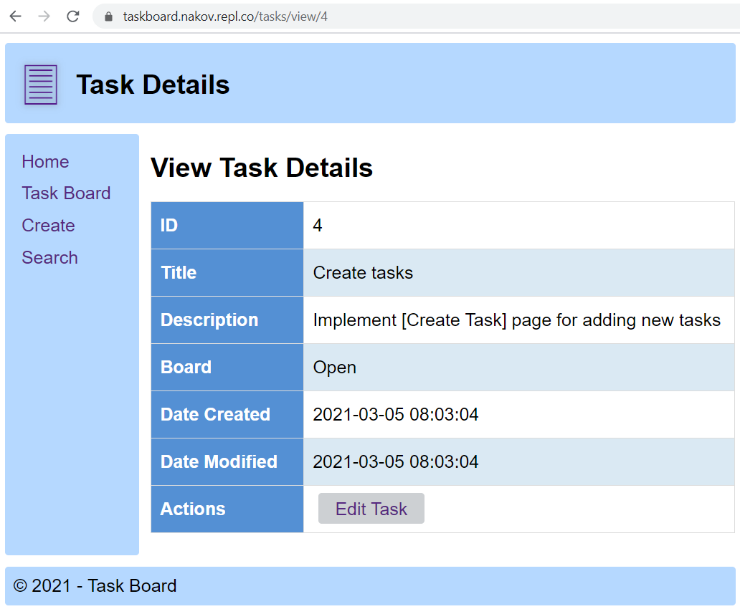
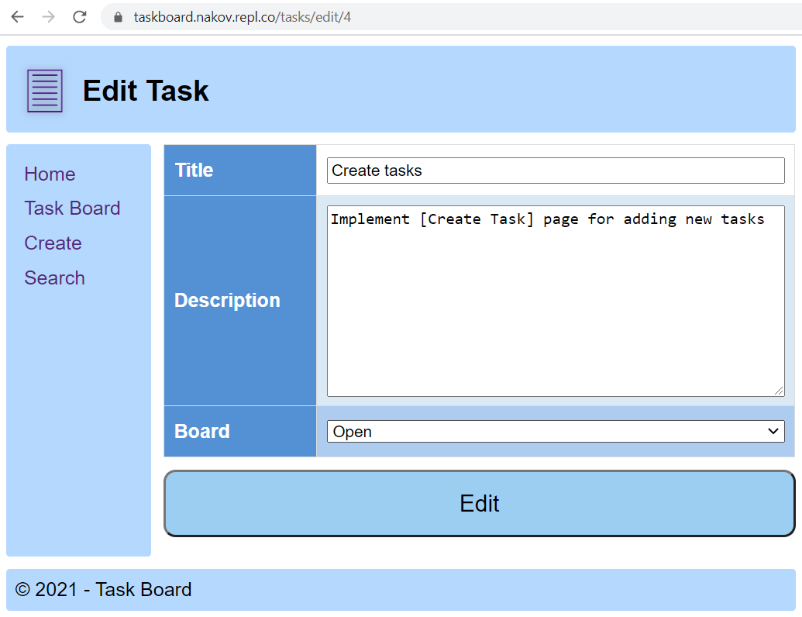
The **"** Task Board**" Web app** supports the following operations:

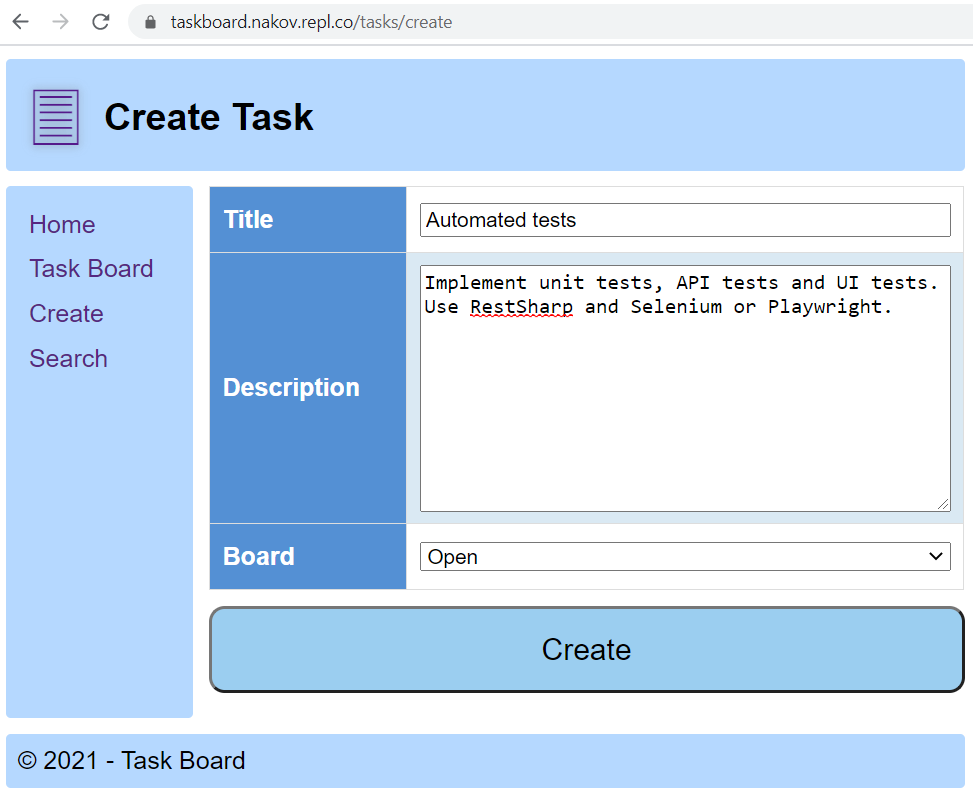
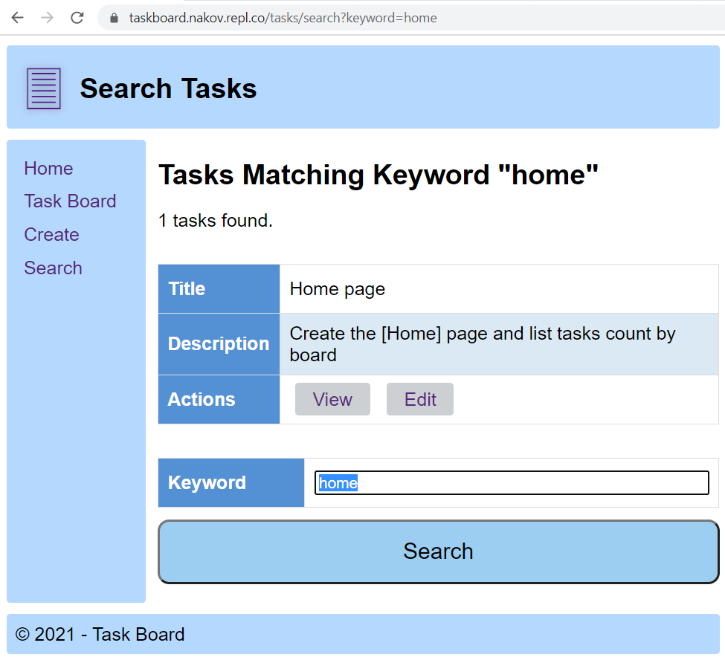
* Home page (view tasks count + menu): /
* View the boards with tasks: /boards
* Search tasks form: /tasks/search
* Search tasks by keyword: /tasks/search/:keyword
* View task details (by id): /tasks/view/:id
* Add new task (title + description): /tasks/create
* Edit task / move to board: /tasks/edit/:id

Run the Web app from: <https://taskboard.nakov.repl.co>.

You can browse the app **source code** and play with the app at: <https://repl.it/@nakov/taskboard>.

[](https://user-images.githubusercontent.com/1689586/110086738-6a320d00-7d9b-11eb-9a59-9fd1ffbab24a.png) [](https://user-images.githubusercontent.com/1689586/110086832-8c2b8f80-7d9b-11eb-9d9c-3d5d94e07f3b.png)

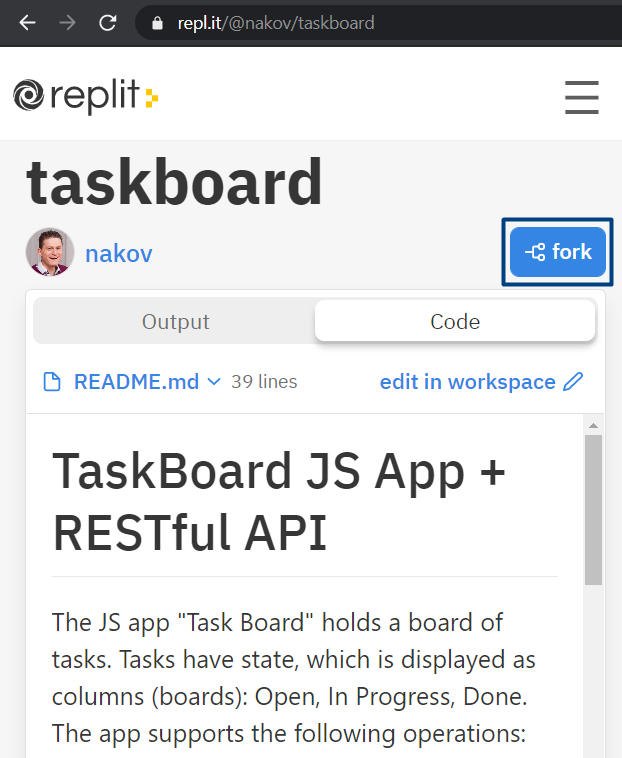
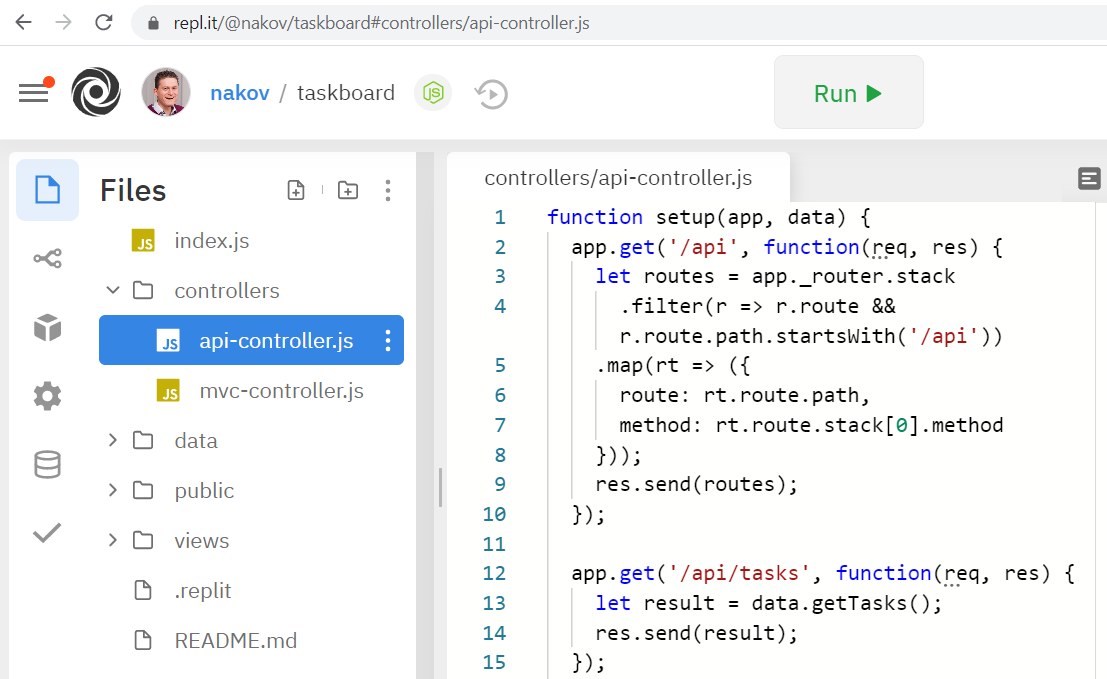
[](https://user-images.githubusercontent.com/1689586/110086878-9a79ab80-7d9b-11eb-97e8-1507e0f90020.png) [](https://user-images.githubusercontent.com/1689586/110086907-a36a7d00-7d9b-11eb-831c-5333992d560b.png)

[](https://user-images.githubusercontent.com/1689586/110087130-edebf980-7d9b-11eb-8307-24c2eb87096d.png) [](https://user-images.githubusercontent.com/1689586/110087188-02c88d00-7d9c-11eb-8fb0-8d9533d72fd2.png)

### Installing and Running the App

To avoid conflicts, it is highly recommended that you **fork the project** for this app from <https://repl.it/@nakov/taskboard> into your **own repl.it account**, e.g.

* [https://repl.it/@{your-account}/taskboard](https://repl.it/@%7byour-account%7d/taskboard)

Optionally, you can **install** and **run** the app on your **local machine**:

|  |
| --- |
| git clone https://github.com/nakov/TaskBoard  cd TaskBoard  npm install  npm start  start http://localhost:8080/api |

### Resetting the App

The app **does not have a persistent database** storage, so you can **reset it** by a simple **restart** (stop & start).

* After restart, you will lose all changes and the default sample data will be populated automatically.

### API Endpoints

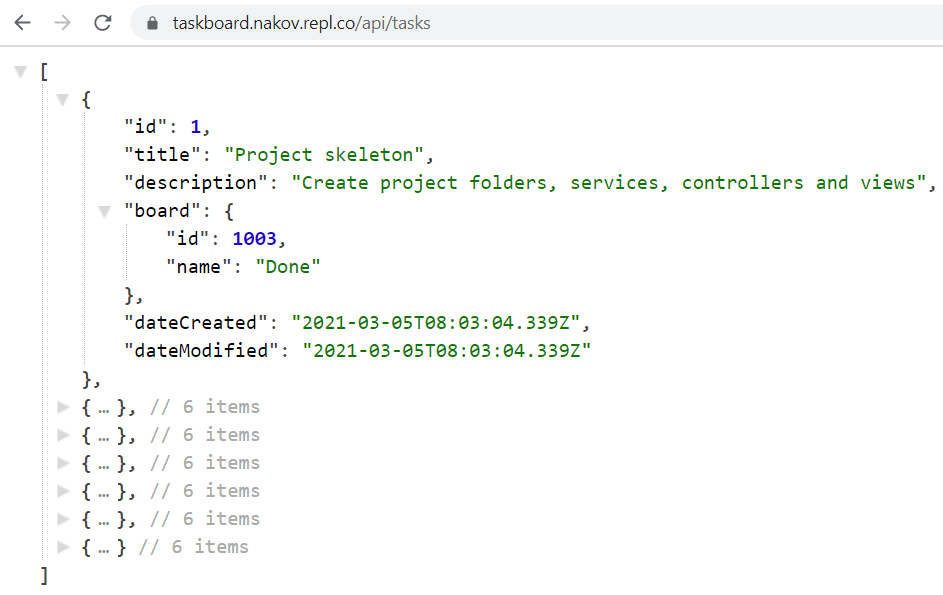
TaskBook exposes a **RESTful API**, available at:

* <https://taskbook.nakov.repl.co/api>

The following endpoints are supported:

* GET /api – list all API endpoints
* GET /api/tasks – list all tasks
* GET /api/tasks/:id – returns a task by given id
* GET /api/tasks/search/:keyword – list all tasks matching given keyword
* GET /api/tasks/board/:boardName – list tasks by board name
* POST /api/tasks – create a new task (post a JSON object in the request body, e.g. {"title":"Add Tests", "description":"API + UI tests", "board":"Open"})
* PATCH /api/tasks/:id – edit task by id (send a JSON object in the request body, holding the fields to modify, e.g. {"title":"changed title", "board":"Done"})
* DELETE /api/tasks/:id – delete task by id
* GET /api/boards – list all boards

This is a sample output from an API call to /api/tasks:

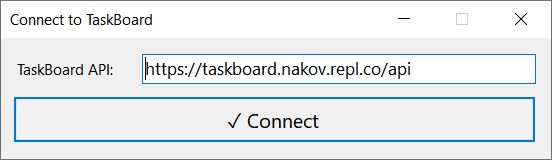


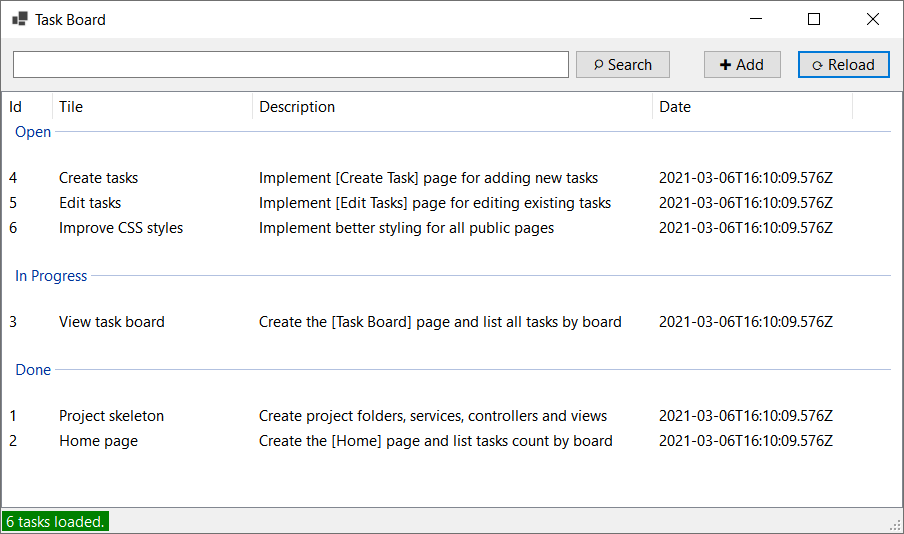
### Windows Desktop Client

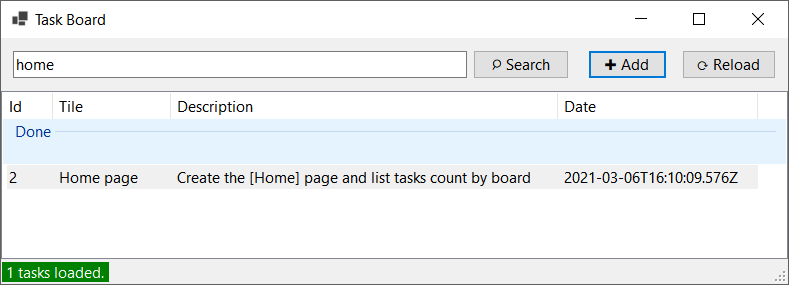
TaskBoard has a Windows desktop client app, available from:

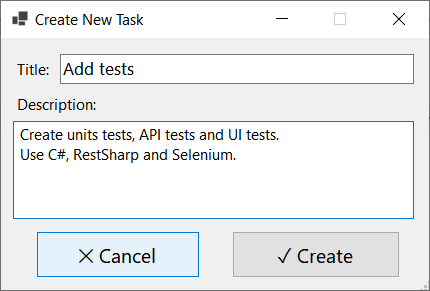
* <https://github.com/nakov/TaskBoard-DesktopClient/releases>

The app **connects** to the TaskBoard API, **displays the task boards**, **searches for tasks** and **creates new tasks**:







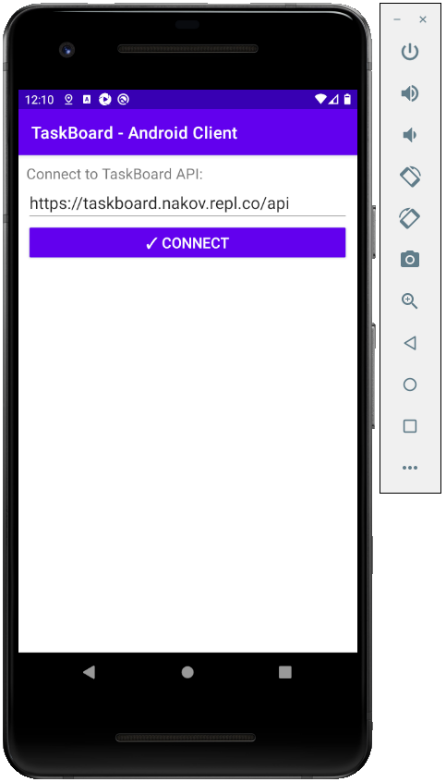
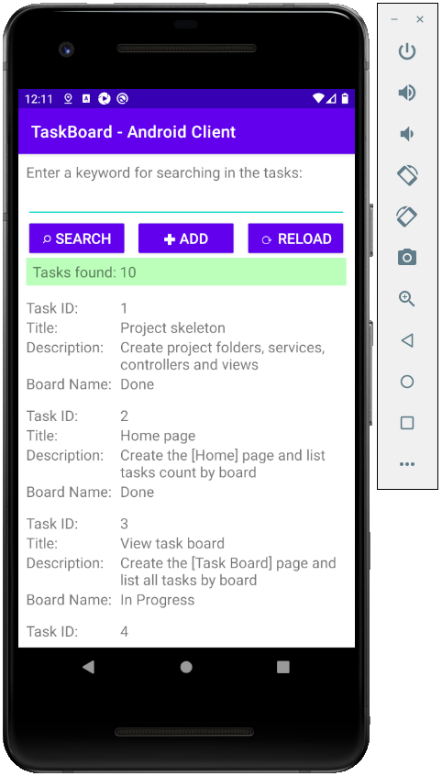
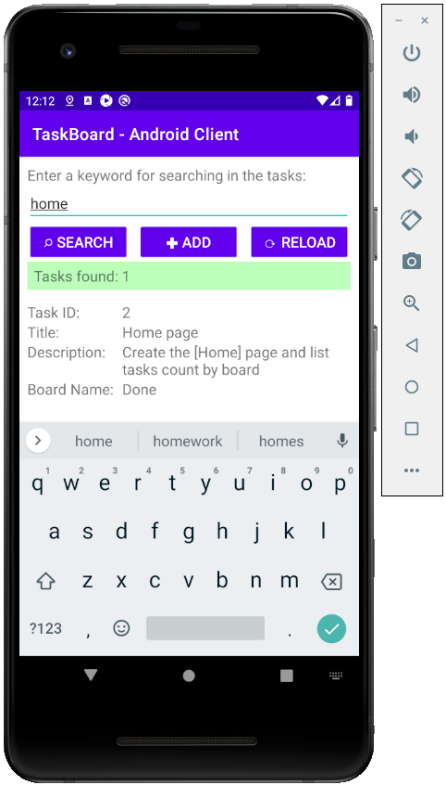
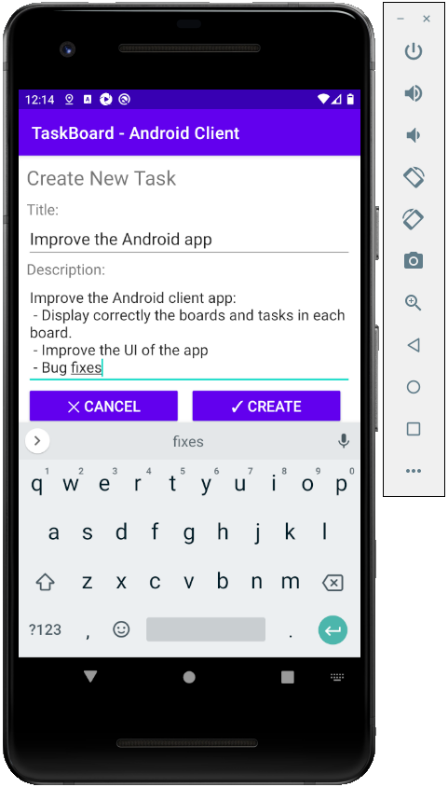


### Android Mobile Client

TaskBoard has an Android mobile client app, available from:

* [https://github.com/nakov/TaskBoard-AndroidClient/releases](https://github.com/nakov/TaskBoard-AndroidClient/releases/)

The app **connects** to the TaskBoard API, **displays the task boards**, **searches for tasks** and **creates new tasks**:

## TaskBoard RESTful API: Automated API Tests

Your task is to write **automated tests** (in C#, Java, JavaScript or other language) for the above provided API endpoints. You should implement the following automated tests:

* **List the tasks** and assert that the first task from board “Done” has title “Project skeleton” (8 score).
* **Find tasks** by keyword “home” and assert that the first result has title “Home page” (5 score).
* **Find tasks** by keyword “missing{*randnum*}” and assert that the results are empty (5 score).
* Try to **create a new task**, holding invalid data, and assert an error is returned (5 score).
* Create a **new task**, holding valid data, and assert the **new task is added** and is properly listed in the task board (12 score).

You are free to use a **testing framework** of choice (e. g. NUnit or JUnit) and **external libraries** (e. g. RestSharp).

## TaskBoard Web App: Automated Selenium UI Tests

Write **Selenium-based automated UI tests** for the “**TaskBoard**” app. You should implement the following **automated UI tests**:

* **List the tasks** and assert that the first task from board “Done” has title “Project skeleton” (8 score).
* **Find tasks** by keyword “home” and assert that the first result has title “Home page” (5 score).
* **Find tasks** by keyword “missing{*randnum*}” and assert that the results are empty (5 score).
* Try to **create a new task**, holding invalid data, and assert an error is returned (5 score).
* Create a **new task**, holding valid data, and assert the **new task is added** and is **properly listed** in the task board (12 score).

You are free to use a **testing framework** of choice (e. g. NUnit or JUnit), but your primary Web UI automation tool should be **Selenium**. You are free to use **external libraries and tools**.

## Appium Tests

Choose one of the next two problems: **Android app UI tests** or **Windows app UI tests**.

## TaskBoard Mobile App: Automated Appium UI Tests

Write **Appium-based automated mobile UI tests** for the “**TaskBoard**” Android mobile app. Implement the following automated testing **scenario** (30 score):

* Open the app.
* Connect to your backend API service.
* Assert the first listed tasks has title “**Project skeleton**”.
* Add a new task (with valid uniquely generated title).
* Search for the new task.
* Assert the results holds the new task.

You are free to use a **testing framework** of choice (e. g. NUnit or JUnit), but your primary Android UI automation tool should be **Appium**. You are free to use **external libraries and tools**.

You are free to implement and run the tests in cloud-based testing platform like **BrowserStack** or in a local instance of Appium with Android Emulator or physical Android device.

## TaskBoard Desktop App: Automated Appium UI Tests

Write **Appium-based automated Windows UI tests** for the “**TaskBoard**” Windows desktop app. Implement the following automated testing **scenario** (30 score):

* Open the app.
* Connect to your backend API service.
* Assert the lists of tasks hold a task with title “**Project skeleton**”.
* Add a new task (with valid uniquely generated title).
* Search for the new task.
* Assert the results holds the new task.