XebiaLabs Developer Assessment

XebiaLabs – xl-assessment@xebialabs.com – Version 1.3, 2015-06-12

XebiaLabs Product Developer Assessment

This is the XebiaLabs product developer assessment. The assessment is intended to give us insight into your technical abilities, development approach and general technical working habits. We view your performance on this assessment as indicative of the work you will deliver as a XebiaLabs developer.

Contents

The assessment consists of:

- an assignment to prepare beforehand
- a presentation about your implementation of the assignment at the XebiaLabs office.

Assignment

The assignment is the "XL Spaceship" case, described below. Read the case carefully and approach it as you would a regular project. The assignment should be implemented in a language of your choice as long as it runs on the JVM. We advise you to pick technology you are familiar with so you can be instantly productive.

Tackle each of the user stories in order, progressing to the next one only when you've finished your implementation. We do not require you to complete all user stories, but the more you manage to do, the better. Each user story contains test cases so you can see if your implementation works. In addition to the test cases mentioned in this document, we will subject your solution to more rigorous automated tests after your submission.

We expect that you'll need about 8 to 10 hours to prepare for the assessment. **Please keep the subject of the assessment confidential so we can use it again in the future**. This means **don't** post it on a publicly accessible site such as github.com.

Presentation

The assessment presentation will be done either in person in the XebiaLabs office or remotely. You are free to deliver your presentation in any form, but we expect you to cover:

- the overall approach you took to the assignment
- the architecture of the solution and the technologies used
- your solution for each of the user stories

Here are some special considerations for the assessment presentation.

XebiaLabs office

- Bring a laptop with a working development environment.
- There will be a beamer present at the office with VGA, DVI and HDMI connectors. Please bring the necessary converters/cables that fit your laptop.

Delivery

Please send the assessment solution to XebiaLabs **one day** in advance of your presentation to allow us time to have a look at it, run it through out automated (REST) test suite and prepare for the assessment.

Sending

XL Spaceship

After having made a start of conquering the world with our Deployment, Release and Test automation software, it is now becoming time to start the conquest of the galaxy. In order to train our salesmen for the challenges that the universe can throw at us, we need to have a spaceship simulator.

We trust that our salesmen can pilot a spaceship, how hard can that be, right? But we need to prepare them for space combat as the competition out there is fierce. Not only from our direct competitors, but also from any non-friendly ETs that one might encounter in outer space. Hence we ask you, the Software Developer, to help us build a *distributed* space battle simulator. It would be great if it also would automatically engage any enemies, without our sales intervening.

Assumptions

- Each XL Spaceship instance is controlled by a single player, but can participate in multiple games at the same time (possibly against the same opponent).

- There are basically 2 APIs, the `protocol` API, which allows to XL Spaceship instances to communicate, and the `user` API which allows interaction with the player. Look carefully at the API definition for each story.
- XL Spaceship instances are not allowed to communicate to each other over the `user` API, only the `protocol` API.
- Unless indicated otherwise the standard HTTP Response Code is `200 (OK)`

Rules

XL Spaceship is played using the following rules:

- The playing board consists of a 16x16 grid, where the columns and rows are numbered in hexadecimal (i.e. from `0` to `F`).
- A shot is indicated as `ROWxCOLUMN`. For instance `1xD`, which indicates that a shot was fired at the cell in row 2, column 14.
- Each player owns (and must place) the following set of spaceships in his grid, one of each.

```
Winger   Angle   A-Class   B-Class   S-Class
* *       *         *        **        **
* *       *        * *       * *      *
 *        *        ***       **        **
* *       ***      * *       * *         *
* *                          **        **
```

- When a game is initiated, positions of the spaceships in the grid are fixed, and they do not change throughout the game.
- Spaceships cannot overlap on the grid.
- Each player is allowed to shoot a salvo of shots equalling the number of spaceships he controls that have not yet been destroyed.
- A spaceship can be placed in any rotation of the form described above, but they cannot be mirrored.

Clarifications

The XL Spaceship product owners are available at xl-assessment@xebialabs.com to answer any questions or clarify any requirements.

User stories

The product owner for XL Spaceship has defined the following user stories for you to implement.

XLSS-1: As a player, I want to receive a new simulation request for a simulation with another player.

This first story is all about setting up a game with another player. On the endpoint we define here, your XL Spaceship instance will be challenged by another XL Spaceship instance. The challenging instance will indicate on which address/port it is reachable for playing the game. The response should contain the unique `game_id` of this match. It also contains the starting player, determined at random. The REST API you need to implement is:

Request: POST /xl-spaceship/protocol/game/new

```json
{
  "user_id": "xebialabs-1",
  "full_name": "XebiaLabs Opponent",
  "spaceship_protocol": {
    "hostname": "127.0.0.1",
    "port": 9001
  }
}
```

Response: Http Status 201 (Created)

```json
{
  "user_id": "player",
  "full_name": "Assessment Player",
  "game_id": "match-1",
  "starting": "xebialabs-1"
}
```

When the game is successfully created, both players *at random* place their ships on the grid and the game can be played.

XLSS-2: As a player, I can be fired upon by my opponent.

The main part of the game is of course that 2 XL Spaceship instances can play against each other. You need to implement the REST API to receive a salvo of shots from your opponent.

Request: PUT /xl-spaceship/protocol/game/<game_id>

JSON
```
{
  "salvo": ["0x0", "8x4", "DxA", "AxA", "7xF"]
}
```

Response

JSON
```
{
  "salvo": {
    "0x0": "hit",
    "8x4": "hit",
    "DxA": "kill",
    "AxA": "miss",
    "7xF": "miss"
  },
  "game": {
    "player_turn": "player-1"
  }
}
```

The status of each shot can be one of the following:

- `hit` : The shot was a hit on a ship
- `kill` : The shot was the last hit on a ship, effectively destroying it.
- `miss` : The shot missed

> Any shot fired upon an already fired upon cell should always result in a 'miss'.

If the game is finished because your last ship was destroyed, return the following response instead:

Response

```
JSON
{
  "salvo": {
    "0x0": "hit",
    "8x4": "hit",
    "DxA": "kill",
    "AxA": "miss",
    "7xF": "miss"
  },
  "game": {
    "won": "xebialabs-1"
  }
}
```

Any subsequent fire request for this game should result in a `404 (Not Found)` Status Code, as it is not allowed to fire any further shots. The response for such a call should look similar like the response above, but all shots should be categorized as misses.

XLSS-3: As a player, I want to fire a salvo of shots to my opponent.

Using the REST API defined in this story, call the REST API defined in XLSS-2 on the opponent's XL Spaceship instance. The user part of the REST API is defined as:

Request: PUT /xl-spaceship/user/game/<game_id>/fire

```
JSON
{
  "salvo": ["0x0", "8x4", "DxA", "AxA", "7xF"]
}
```

Response

```
JSON
{
  "salvo": {
    "0x0": "hit",
    "8x4": "hit",
    "DxA": "kill",
    "AxA": "miss",
    "7xF": "miss"
  },
  "game": {
    "player_turn": "player-1"
  }
}
```

XLSS-4: As a player, I want to view the status of the game in progress

Implement the viewing of your own space grid, and what you know of your opponent's grid. On your own grid, you should see all your spaceships. The REST API to implement is:

Request: `GET /xl-spaceship/user/game/<game_id>`

Response

```
JSON
{
  "self": {
    "user_id": "player-1",
    "board": [
      "**.**...........",
      "..*.............",
      "**.**....***....",
      "..........*.*...",
      ".........***....",
      "................",
      "................",
      "............***.",
      ".......**.....*.",
      "......*.*.....*.",
      ".......**.....*.",
      "......*.*.......",
      "...*...**.......",
      "*.*.*...........",
      "*.*.*...........",
      ".*.............."
    ]},
  "opponent": {
    "user_id": "xebialabs-1",
    "board": [
      "................",
      "................",
      "................",
      "................",
      "................",
      "................",
      "................",
      "................",
      "................",
      "................",
      "................",
      "................",
      "................",
      "................",
      "................",
      "................"
    ]},
  "game": {
    "player_turn": "player-1"
  }
}
```

The following symbols should be used in the grid:

- `*` : A quadrant taken by part of a ship which has not been hit yet.
- `-` : A quadrant that contains a missed shot

- `X` : A quadrant taken by part of a ship which was hit by a shot.
- `.` : An empty or unknown quadrant.

> When the game has finished, the `game` part of the JSON will not contain the `player_turn` property, but rather the `won` property, see also XLSS-2

XLSS-5: As a player, I want to have an *autopilot* which runs the simulation for me.

Implement a (simple) autopilot that automatically fires back when it's the player's turn. So when you receive a salvo, respond to that, and then fire your own salvo. Using the following REST API it should be possible to turn on the autopilot for a specific game. The autopilot should automatically stop once the game is lost or won.

Request: `POST /xl-spaceship/user/game/<game_id>/auto`

Response: HTTP Status Code: `200 (OK)`

XLSS-6: As a player, I want to play via an user interface, instead of through REST requests.

Implement a UI on top of the game that visualizes the game in progress, allowing the user to play the game through the UI.

XLSS-7: As a player, I want to configure another set of rules for a game.

Add to the request of XLSS-1 an *optional* field called `rules`, with a string value. The value can be one of the following:

- `standard` : The default set of rules described above.
- `X-shot` : A player is allowed a salvo of `X` shots, where `X` is a number between `1` and `10`. There is no deduction for lost ships. For instance `4-shot`
- `super-charge` : If you destroy a ship this turn, get awarded another salvo of shots.
- `desperation` : Start with 1 shot salvos and for each of your ships destroyed, you are allowed an extra shot per salvo.

The new-game response should indicate that the intended rule-set is taken into account using the same optional `rules` field, eg.

Request: POST /xl-spaceship/protocol/game/new

```json
{
  "user_id": "xebialabs-1",
  "full_name": "XebiaLabs Opponent",
  "rules": "6-shot",
  "spaceship_protocol": {
    "hostname": "127.0.0.1",
    "port": 9001
  }
}
```

Response: Http Status 201 (Created)

```json
{
  "user_id": "player",
  "full_name": "Assessment Player",
  "game_id": "match-1",
  "starting": "xebialabs-1",
  "rules": "6-shot"
}
```

XLSS-8: As a player, I want to challenge another player to a game.

Implement the calling side for a new game. The REST API defined for this is:

Request: POST /xl-spaceship/user/game/new

```json
{
  "spaceship_protocol": {
    "hostname": "10.10.0.2",
    "port": 9000
  },
  "rules": "super-charge"
}
```

Success Response

```
Status: 303 (See Other)
Location: /xl-spaceship/user/game/<game_id>
Content-Type: text/html

A new game has been created at xl-spaceship/user/game/<game_id>
```

Any non-success response (ie. unreachable XL Spaceship instance) should result in a `400 (Bad Request)` status code.

The sample data shown will challenge the XL Spaceship instance at `10.10.0.2:9000` on the REST API defined in XLSS-1.

Version 1.3
Last updated 2015-06-12 10:17:24 CEST