Author information
Name: Lucassen, Mario M.
Course: Software Engineering
Email: M.lucassen@student.fontys.nl

PAZURU

SOFTWARE ARCHITECTURE DOCUMENT

Document no: 3
Version: 1.02
File: Pazuru Software Architecture Document.docx
Created: 12.05.2019
Last changes: 12.06.2019

Version history

Version	Date	Changes	Author[1]
0.01	12.05.2019	Document created with Introduction, Architectural goals and constraints, Architectural representation, System context, Containers and chosen technologies, Components, and Class diagrams.	Lucassen, Mario M. M.lucassen@student.fontys.nl
1.00	12.05.2019	Final version of document.	Lucassen, Mario M. M.lucassen@student.fontys.nl
1.01	20.05.2019	Added interface specification.	Lucassen, Mario M. M.lucassen@student.fontys.nl
1.02	12.06.2019	Fixed interface specification.	Lucassen, Mario M. M.lucassen@student.fontys.nl

[1] Author's name with email address

Table of contents

Table 1. Global definitions.

Name	Description
Enigmatologist	Someone who studies and writes mathematical, word or logic puzzles.

1. Introduction

This introduction provides an overview of the entire *Software Architecture Document* for the Pazuru puzzle platform. It includes the purpose, scope, author profile, and an overview of the document.

1.1. Purpose

The primary purpose of the Pazuru puzzle platform is for Enigmatologists to satisfy their puzzling needs. This document provides an architectural overview of the Pazuru puzzle platform. This document is also intended to capture and convey the significant architectural decisions which have been made in designing the system. In a way that that other software architects can better understand the project being build and what problems are being solved.

1.2. Scope

The scope of the product being made is the Pazuru puzzle platform with its subsystems, Sudoku and Hitori.

Scope includes
Puzzle platform
Sudoku (subsystem)
Hitori (subsystem)

1.3. Author Profile

The author of the puzzle platform is Lucassen, Mario M, a Software Engineering student at Fontys. The product will be developed at the Fontys University of Applied Sciences, Rachelsmolen 1 in Eindhoven during the time allotted for the Software Engineering course.

Author information			
Name:	Lucassen, Mario M.	Course:	Software Engineering
Email:	M.lucassen@student.fontys.nl	Role:	Software Developer

1.4. Overview

This document consists of 7 sections, which are described below:

- Section 1 is an introduction to the software architecture of the Pazuru puzzle platform.
- Section 2 addresses the goals and constrains of the system's architecture.
- Section 3 describes the architectural representation of the system.
- Section 4 describes the context of the system.
- Section 5 describes the containers and chosen technologies.
- Section 6 describes the components of the system.
- Section 7 describes class diagrams of the system being build.

2. Architectural goals and constraints

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

1. The system being build is meant as a platform for several puzzles to be added to as subsystems. This means that there needs to be an interface in which all puzzles constraints can be communicated through. Therefore one of the primary stakeholders in this document and the system as a whole are the future architects and designers, not necessarily users as would normally be the case. As a result, one goal of this document is to be useful to future architects and designers to add new puzzles to the platform.

2. The system will be build using Microsoft .NET technologies and will use an open source RDBMS (MYSQL) for data persistence and can be deployed on both Windows servers and Linux webservers. These special deployment requirements require additional consideration in the development of the architecture.

3. Section 2 of the *Software Analysis Document* outlines a number of rules, actions and quality attributes the system needs to adhere. The architecture seeks to do this through the use of modularization and testing the system thoroughly in the *System Test Plan Document*.

3. Architectural representation

This document details the architecture using the C4 model, invented by Simon Brown[2]. The diagrams that will be used to convey the system being build are described below.

A **System Context Diagram (C1)** is a diagram that is able to convey the big picture of the product being build, with a focus on people and software systems rather than technologies, protocols and other low-level details.

A **Container Diagram (C2)** is a diagram that is able to convey a the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another.

A **Component Diagram (C3)** is a diagram that shows the more technological side of containers. It shows that containers are made up of several “components”, what each of those components are, their responsibility and the technology/implementation details.

A **Class Diagram (C4)** is a diagram that shows how the implementation of the components are designed. Most of the time the more important or complex classes are shown in a class diagram, but this depends on the architect designing the software.

[2] Simon Brown, inventor of the C4 model for Software Architecture: https://simonbrown.je/

4. System context (C1)

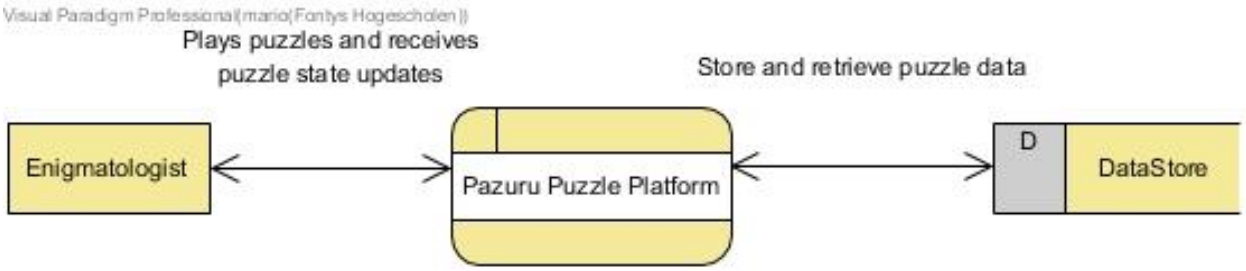

Figure 1. Pazuru System Context Diagram.

The figure above shows the System Context Diagram. It shows the context of the software. The system being build is shown as a Process, a user (Enigmatologist) of the system is shown as an Entity and the storage of the system is shown as a DataStore. This figure is made with Visual Paradigm.

5. Containers and chosen technologies (C2)

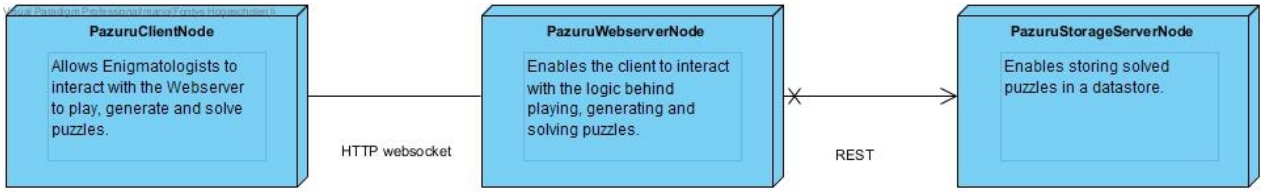

Figure 2. Pazuru Container Diagram.

The figure above shows the Container Diagram. In this diagram 3 different nodes are involved with the distributed puzzle platform: PazuruStorageServerNode, PazuruWebserverNode and PazuruClientNode. The communication between the PazuruWebserverNode and the PazuruStorageServerNode uses a RESTful API. The communication between the PazuruClientNode and the PazuruWebserverNode uses Websockets. This figure is made with Visual Paradigm.

6. Components (C3)

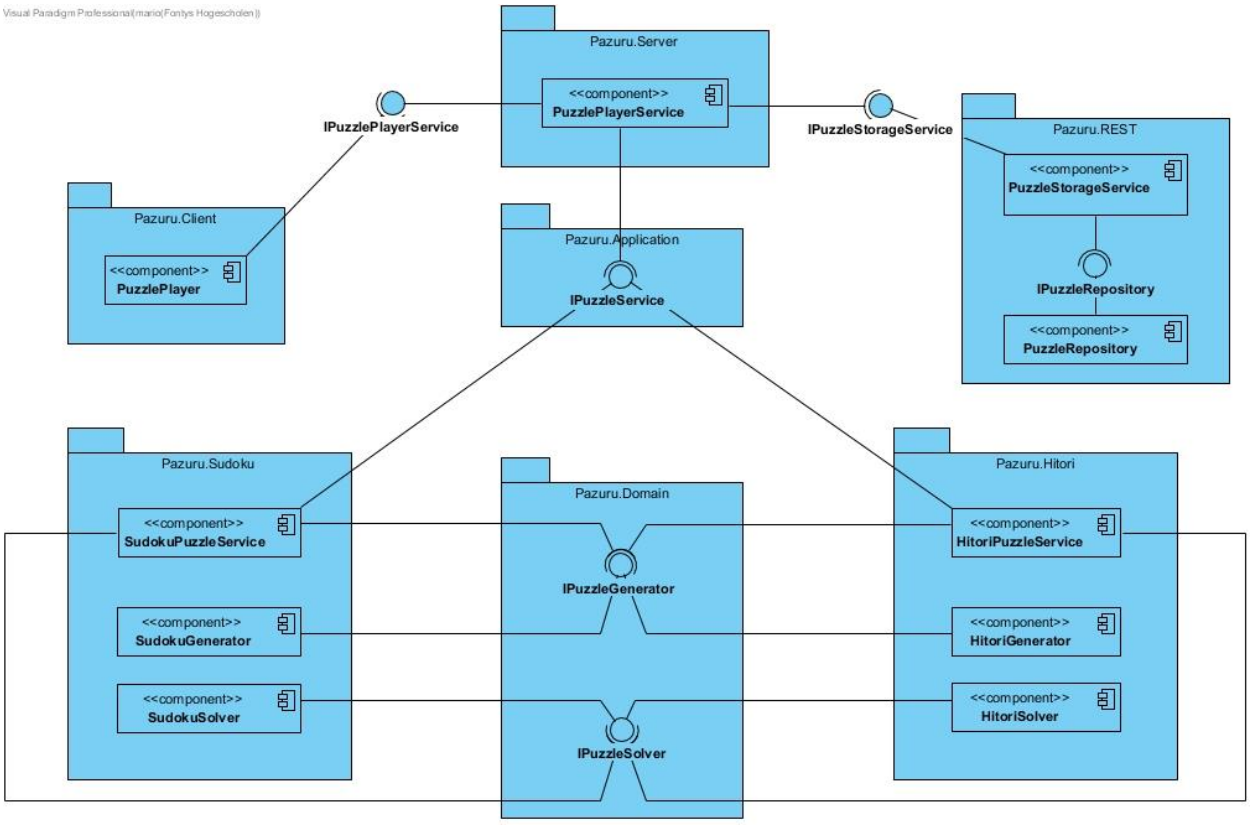

Figure 3. Pazuru Component Diagram.

The figure above shows the Component Diagram. In this diagram it can be observed that the server has a shared IPuzzleService interface for the abstraction of the puzzle logic. The IPuzzlePlayerService interface is used to communicate to the server. The IPuzzleStorageService interface is used to communicate with the RESTful API for storing puzzles. The PuzzleStorageService then in turn uses the IPuzzleRepository interface to communicate with the PuzzleRepository. The PuzzleRepository then actually saves the puzzles in a database. This figure is made with Visual Paradigm.

Each puzzle in the platform must have its own implementation of IPuzzleService, IPuzzleGenerator and IPuzzleSolver. The logic for in these components have been described in the Puzzle Specification Documents.

7. Class diagrams (C4)

To keep it simple, the most important packages are shown. This way it is easier for another architect to use this document to create his/her subsystem (puzzle implementation).

7.1. Domain

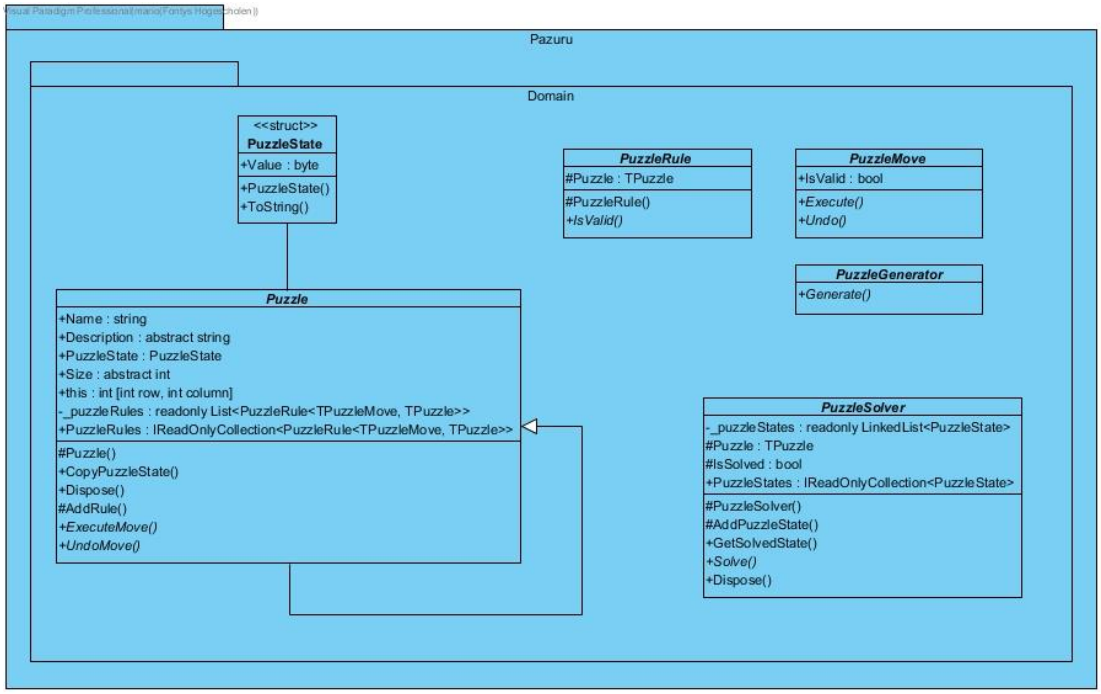

Figure 4. Pazuru Domain Class Diagram.

7.2. Application

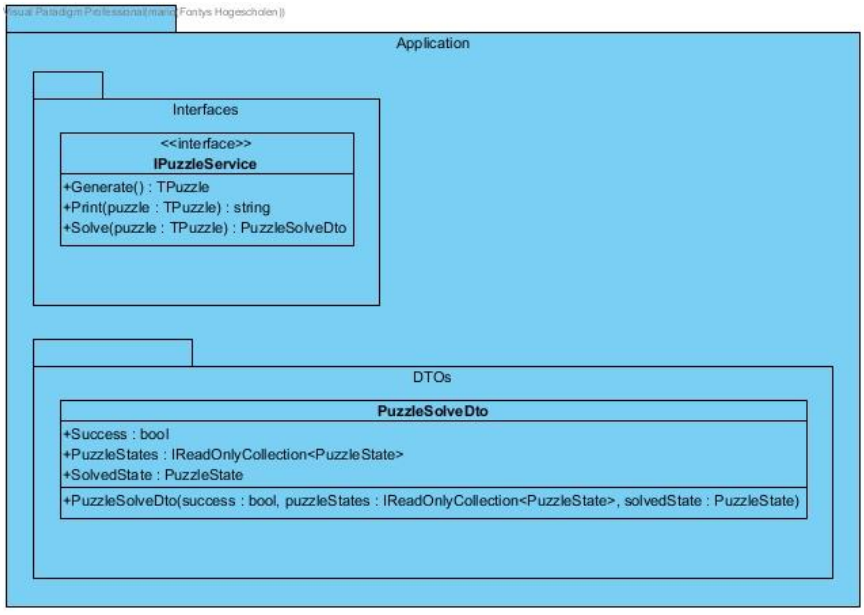

Figure 5. Pazuru Application Class Diagram.

7.3. Sudoku

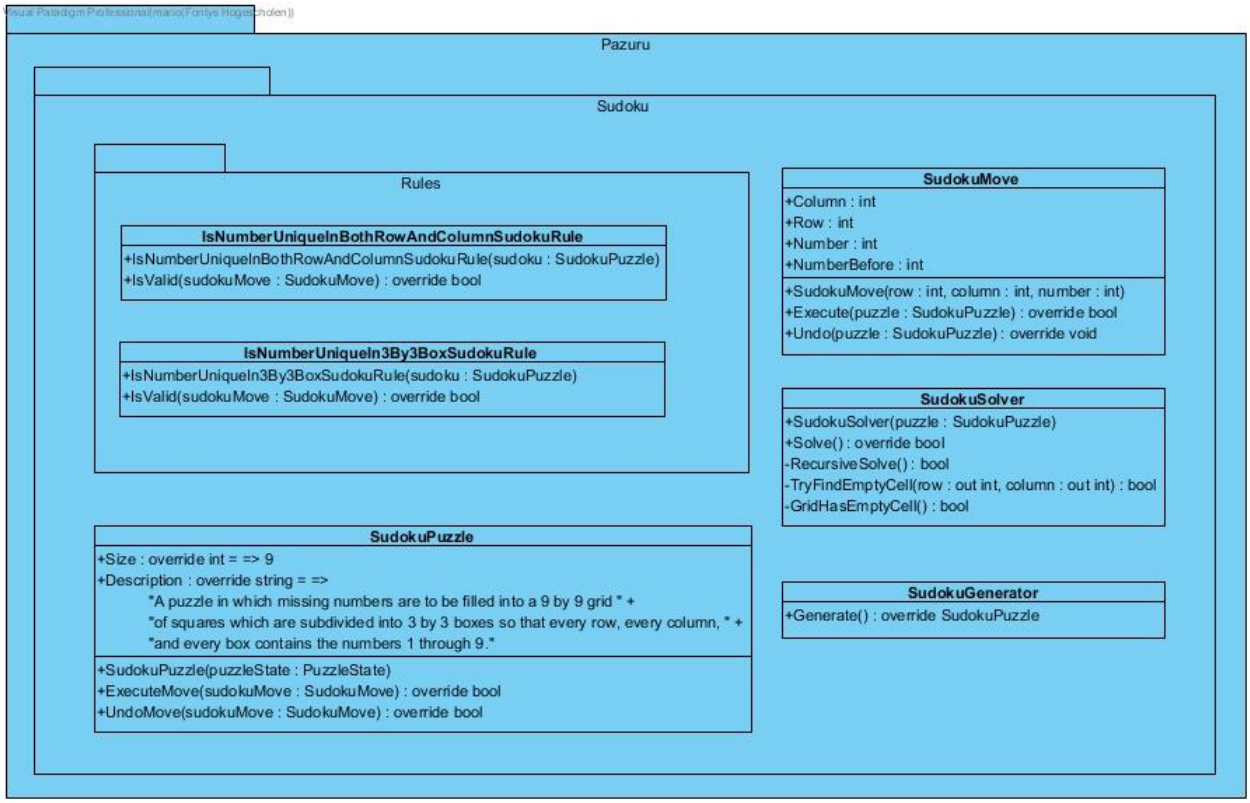

Figure 6. Pazuru Sudoku Class Diagram.

7.4. Hitori

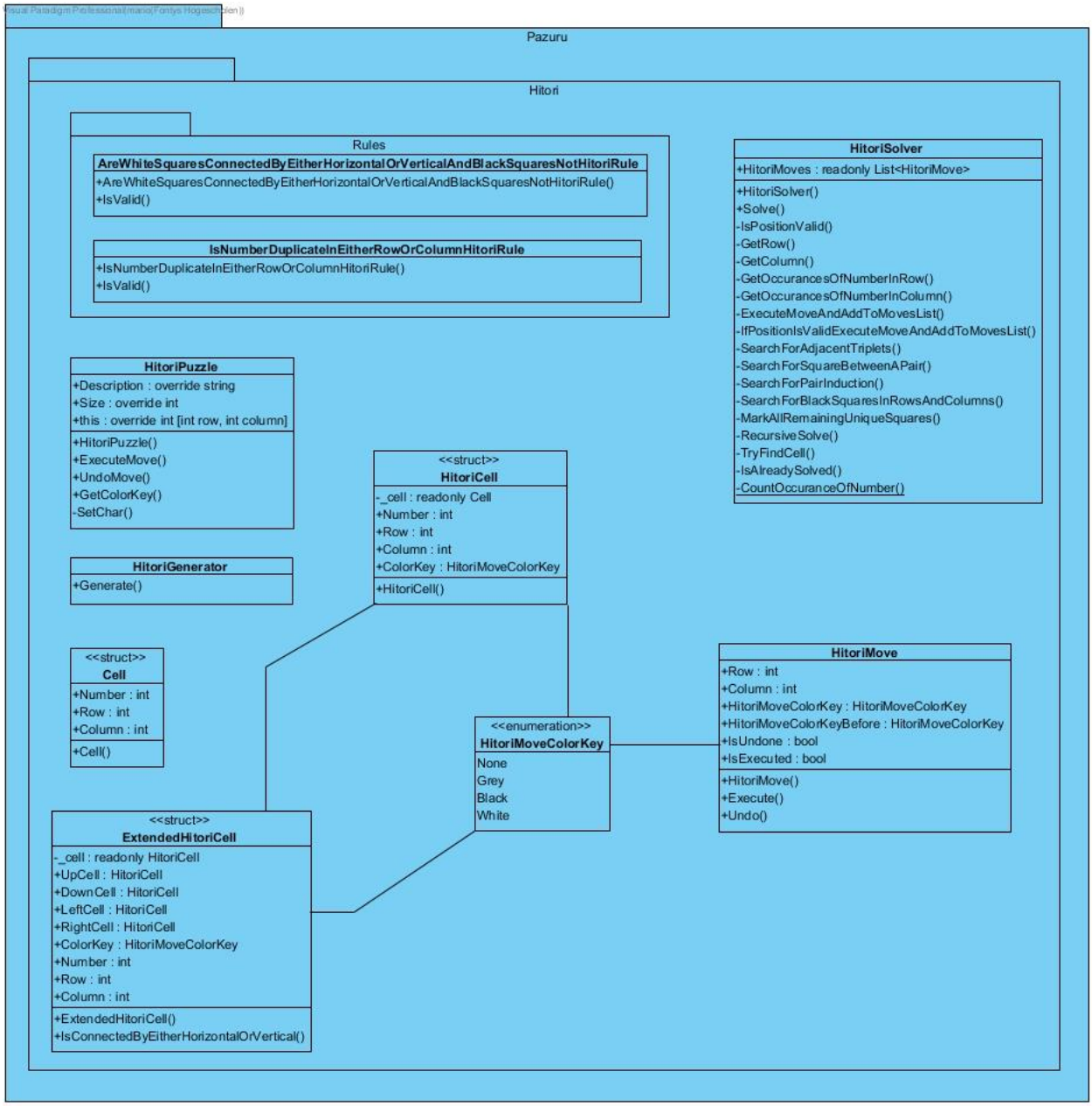

Figure 7. Pazuru Hitori Class Diagram.

8. Interface specification

This chapter contains the interface specification for communication between components.

8.1. Client to server

The client uses websockets with messages in the JSON format to communicate with the server.

The endpoint the client uses to communicate is **wss://localhost:44399/puzzle** every message send to the server has a property called “eventName”, this way the server knows where to delegate the request to.

Generating a sudoku puzzle request

{"eventName":"sudokuGeneratePuzzleRequest"}

Example response:

{"eventName":"sudokuGeneratePuzzleRequest","data":{"puzzleAsString":"034007008080065000000300070200000700710040096005000010500020000001700606009004 30"}, “success”: true}

Solving a sudoku puzzle request

A solve request will send back every single state in solving a Sudoku puzzle.

{"eventName":"sudokuSolvePuzzleRequest","data":{"asString":"034007008080065000000300070200000700710040096005000010500020000001700606009004 30"}}

Example response:

{"eventName":"sudokuPuzzleStateChange","data":{"index":34,"numberAfter":0,"numberBefore":8,"changed":true,"lastEvent":false}, “success”: true}

Get all previously solved puzzles request

{ "eventName": "getAllPreviouslySolvedPuzzlesRequest" }

Example response:

{ "eventName": "getAllPreviouslySolvedPuzzlesRequest", "data": { "puzzles": [{ "sudokuPuzzleState": { "original": "034007008080065000000300070200000700710040096005000010500020000001700606009004 30", "solved": "534297618187465329962381574246819753718543296395726841459632187823174965671958432" } }] } , “success”: true}

8.2. Server to RESTful api

The server uses RESTful https with the messages in the JSON format to communicate with the api.

Saving a solved puzzle state

POST **https://localhost:8090 /puzzles/savePuzzle**


```
{
    "puzzleType": "Sudoku",
    "originalPuzzle": "034007008080065000000300070200000700710040096005000010500020000001700606009004 30",
    "solvedPuzzle": "534297618187465329962381574246819753718543296395726841459632187823174965671958432"
}
```


Example response:


```
{
    "message": "Saved successfully.",
    "success": true,
    "data": {
        "puzzleId": 1,
        "puzzleType": "Sudoku",
        "solvedPuzzle": "534297618187465329962381574246819753718543296395726841459632187823174965671958432",
        "originalPuzzle": "034007008080065000000300070200000700710040096005000010500020000001700606009004 30",
        "_links": {
            "self": {
                "href": "http://localhost:8090/puzzles/sudoku/{id}",
                "templated": true
            }
        }
    },
    "_links": {
        "self": {
            "href": "http://localhost:8090/puzzles/savePuzzle"
        }
    }
}
```


Get all solved puzzles

GET **https://localhost:8090/ puzzles/previouslySolvedPuzzles**


```
{
    "message": "Successfully found all previously solved puzzles.",
    "success": true,
    "data": [
        {
            "puzzleId": 1,
            "puzzleType": "Sudoku",
            "solvedPuzzle": "534297618187465329962381574246819753718543296395726841459632187823174965671958432",
            "originalPuzzle": "034007008080065000000300070200000700710040096005000010500020000001700606009004 30",
            "_links": {
                "self": {
                    "href": "http://localhost:8090/puzzles/sudoku/{id}",
                    "templated": true
                }
            }
        }
    ],
    "_links": {
        "self": {
            "href": "http://localhost:8090/puzzles/previouslySolvedPuzzles"
        }
    }
}
```