Readme

TEAM 2

Contents

Intro **3**
- Overview . . . 3
- Audience and Document Scope . . . 3
- Repository Structure . . . 3

Application Structure **3**
- Application Distribution . . . 4

Communication Structure **4**

Deployment Structure **5**

Footnotes **6**

Intro

Overview

This is Team 2's project as part of the CS3305 module in 2018, which is an online version of Monopoly. You can access a live version of this application here.

Audience and Document Scope

This document is part of the larger documentation associated with this project. It is intended to give a high-level overview of the project, targeted at developers new to the project, and at developers currently involved.

For a guide on how to use the application, targeted at end-users, see the usage document.

For a more detailed guide on how to get involved and contribute to the project, see the contributing document.

Repository Structure

The repository is broken down into two sub-sections:

- frontend: client-side code
  - This is structured as a node package. For more info, see the frontend readme.
- backend: server-side code
  - This is structured as a python package. For more info, see the backend readme.

Application Structure

The application is structured as a standard web application – game state is stored on a server, to which players' clients (browsers) connect via HTTP.

Application Distribution

1. A client requests the initial page from the server via HTTP.
2. The server responds with the page, which includes a javascript program.
3. This javascript program handles client interaction and communicates with the server, and updates the client's screen in response to changes.

Communication Structure

The following diagram shows a component & connector view of the system:

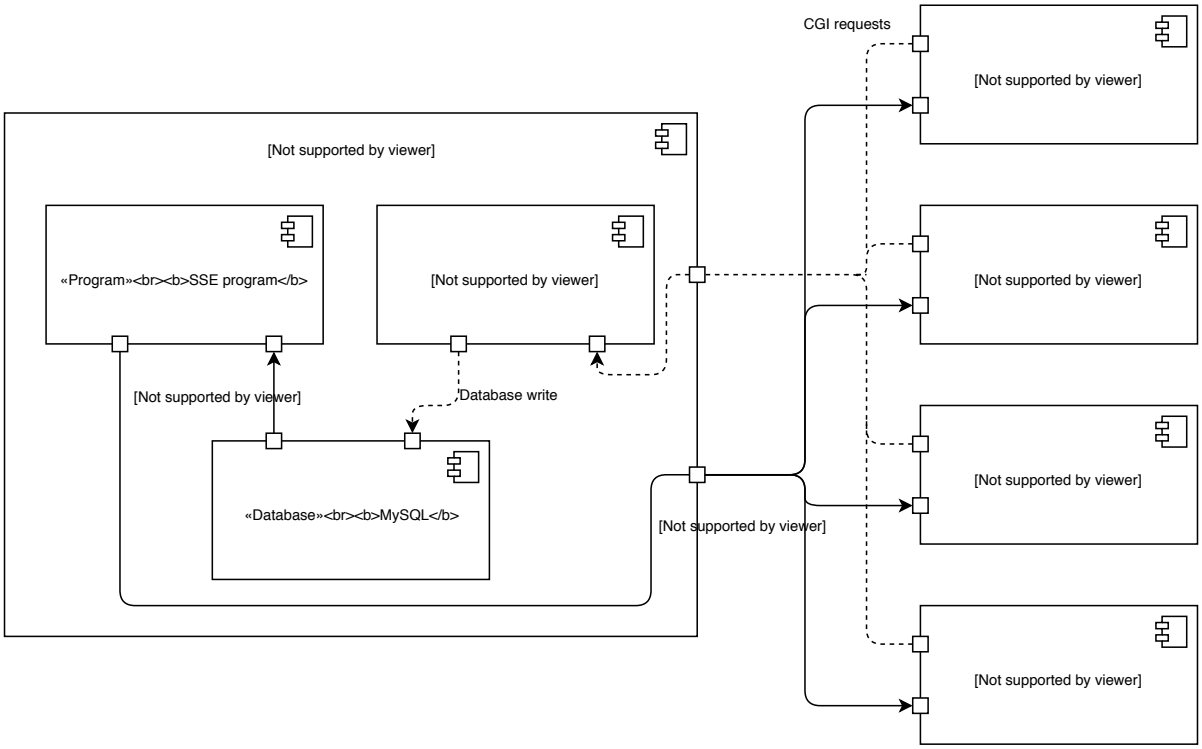

Figure 1: Component & Connector View

There are two main information flows:

1. Clients send requests to apache (represented by dashed lines in the diagram), which runs the corresponding CGI scripts. These scripts interact with a database and send back a response.
2. Clients also initialise a Server-sent Events event stream (represented by a solid line in the diagram). This starts a script running on the server which polls the database for changes – it sends

events back to the client for any changes it finds.

Since the SSE script1 is constantly polling, most CGI scripts don't require a response (except possibly an acknowledgement), and so mostly write to the database, rather than reading from it.

Deployment Structure

The following diagram shows a deployment view of the project:

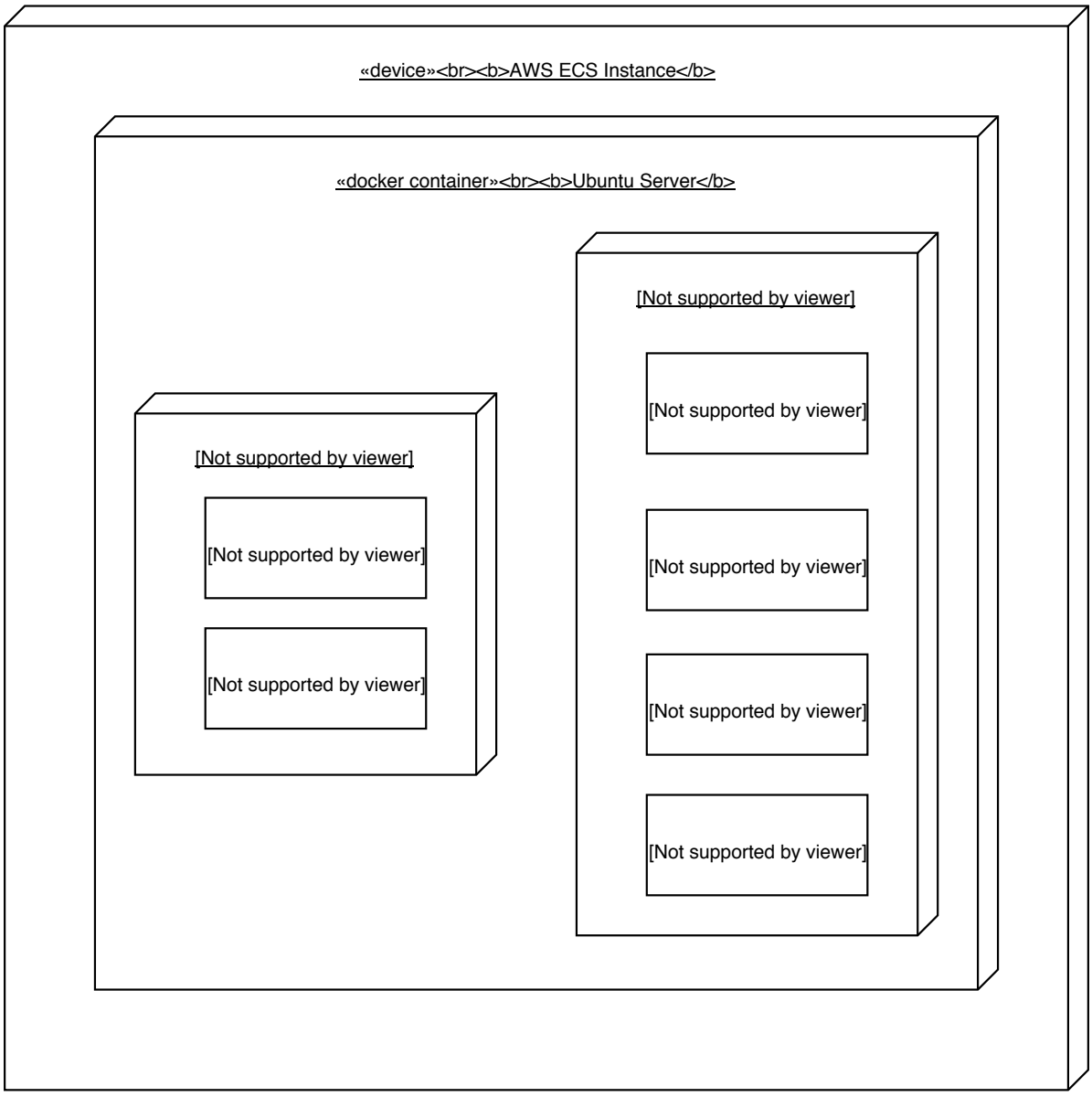

Figure 2: Deployment View

Footnotes

1: This is slightly misleading, as it suggests that there is only one SSE program running. Actually, each client gets its own thread, but they are all running the same program. The client can pass data (e.g. an id) to the SSE script through the query string so that it can behave differently for each client. back