WinnerDrinks

A social drinking game 😡

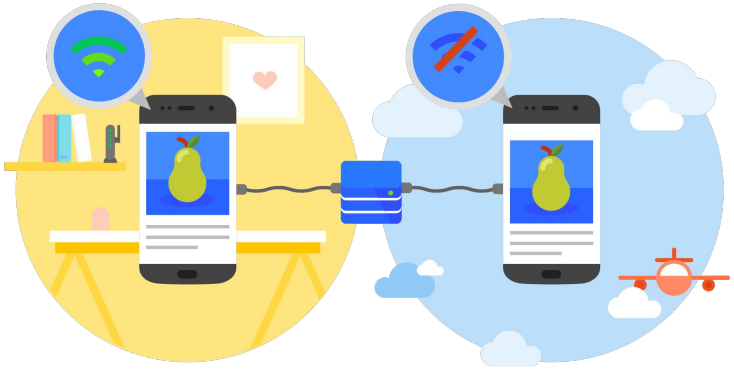
Project overview

- Winner Drinks the application
- Problem /solution
- Project methods / processes

Winner Drinks the application

WinnerDrinks is a **fun** and **social** drinking game.

- ➔ Play to win rather than play to be punished.
- ➔ A game that is built from minigames.
- ➔ An app that can serve a whole group of people; only one user needs to start the app.
- ➔ Available as a progressive web application
  - ◆ A user can get immediate access to the app.
  - ◆ A user can use it offline and online.
  - ◆ A user can add it to the home screen on a smartphone or reading tablets.
  - ◆ Software updates are pushed immediately to production.



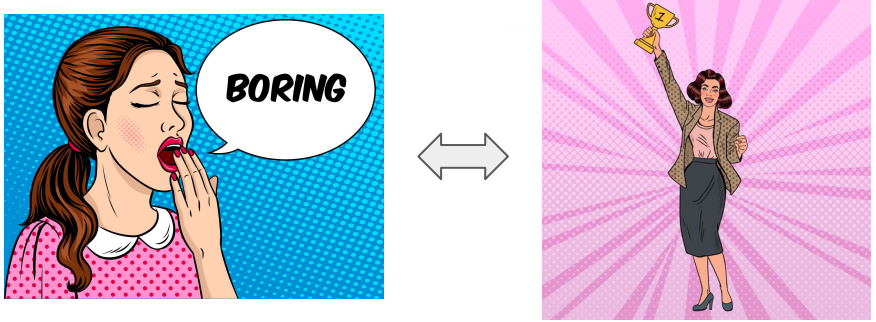
Solution / Problem

Existing games

- Lack of participation
- Loss of interest
- No future usage

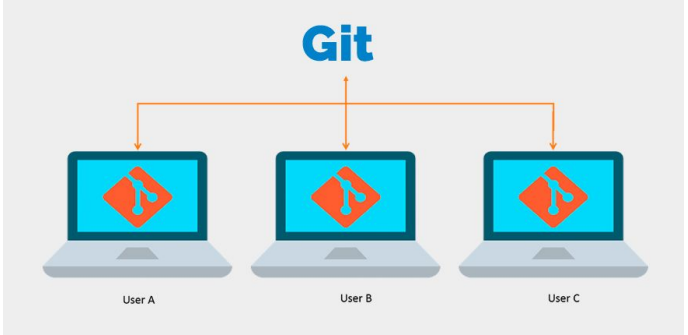
Our solution

- Brings people together
- Keep interest in game
- More interactive



Project methods and processes

- Weekly meetings (Monday)
- Communication channel - Slack
- Version management of the application using Git
- Kanban board
- Living documentation
- Agile approach is a winning concept



Requirements Document: Introduction

- Introduction to both developers and non-developers
- References to requirements that provide a good overview of required functionality
- Definitions and abbreviations, game module

Requirements Document - Document structure

- Application requirements
- Game module requirements
- Templates
- Tables
- Test-cases

Requirements Document: System-wide requirements

2.3.10 Game modules extensibility

Identifier: WD.NF.2

Description: The development of current and future game modules in the application must be able to take place independently of each other .

Affected requirements:

1. *2.4 Module Party Requirements*
2. *2.5 Module Trivia (Multiple-choice) Requirements*
3. *2.6 Module Spin the Wheel Requirements*
4. *2.7 Module Back to Back*

Constraints:

1. There must not be any coupling and dependencies between the game modules.

Priority: Essential

Risk: Medium implementation risk, medium testability risk.

Requirements Document: Use cases

Use Case 2 - Choose game module

Description: A user should be able to disable the by default included game modules.

Actors: A User.

Pre-condition: The application has been started.

Post-condition: The inclusion status of the game modules is updated.

Basic flow:

1. The application displays the start page.
2. The user clicks on the setting menu.
3. The application displays the game modules implemented in the application and a checked checkbox if the game module is included or unchecked if excluded.
4. The user selects exclude or include for one or several game modules and then close the settings menu.

Alternative flow:

4.1 The user returns to the previous page without updating the game modules.

1. The user closes the settings menu.

4.2 Only two game modules are included

1. The user tries to uncheck one of the two included game modules.
2. The application does nothing.

Figure 2: Use case diagram Choose game module

Requirements Document: System interfaces

- Color palette and fonts
- One way navigation
- GUI interface
- Consistency
- External systems/devices

Segoe UI

Times New Roman

Requirements Document: Domain rules

- "Drink responsibly” message
- Minimum/maximum number of players
- Overall flow of the game

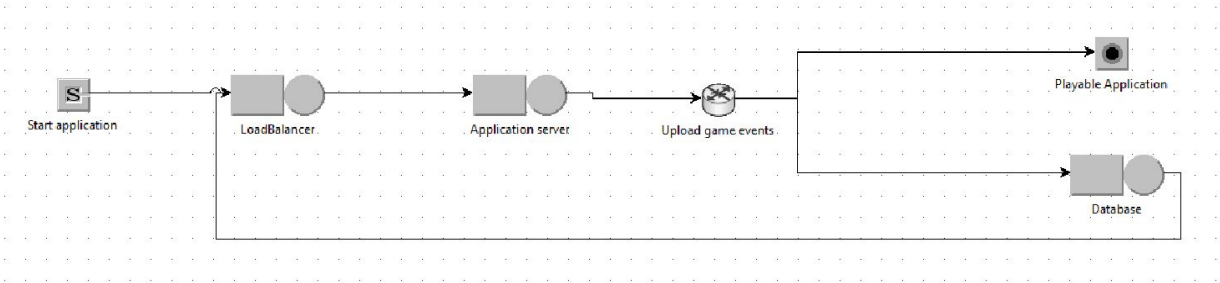
“Warning!
Drink responsibly.
Never drink and drive.
Never drink if you are underage.
Never drink if you are pregnant”.

Requirements Document: System constraints

- Documentation in English
- Application language English
- Documentation available to client
- Source code repository available
- Release 2021-05-30

Requirements Document: Performance modeling

- Assumptions of initial values from 2DV608
- Actual values = more accurate results
- Number of database instances: 1 - 8
- Number of application server instances: 1 - 10



Design

Technologies

- **Programming language:**
  - TypeScript
- **Software development stack, MERN:**
  - MongoDB
  - Express
  - React
  - Node.js
- **Web browser technologies and APIs:**
  - IndexedDB
  - Cache storage
  - Service worker
- **Container tool:**
  - Docker and docker-compose

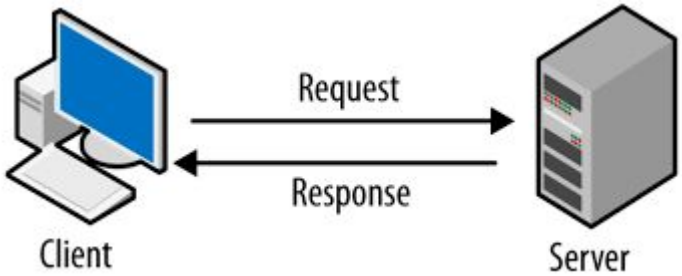
Client-server architecture

Server responsibility

- ➔ Serve the client
- ➔ Serve persistent data to the client

Client responsibility

- ➔ Implement the GUI
- ➔ Interact with the user
- ➔ Application logic and domain rules
- ➔ Support offline use of the app



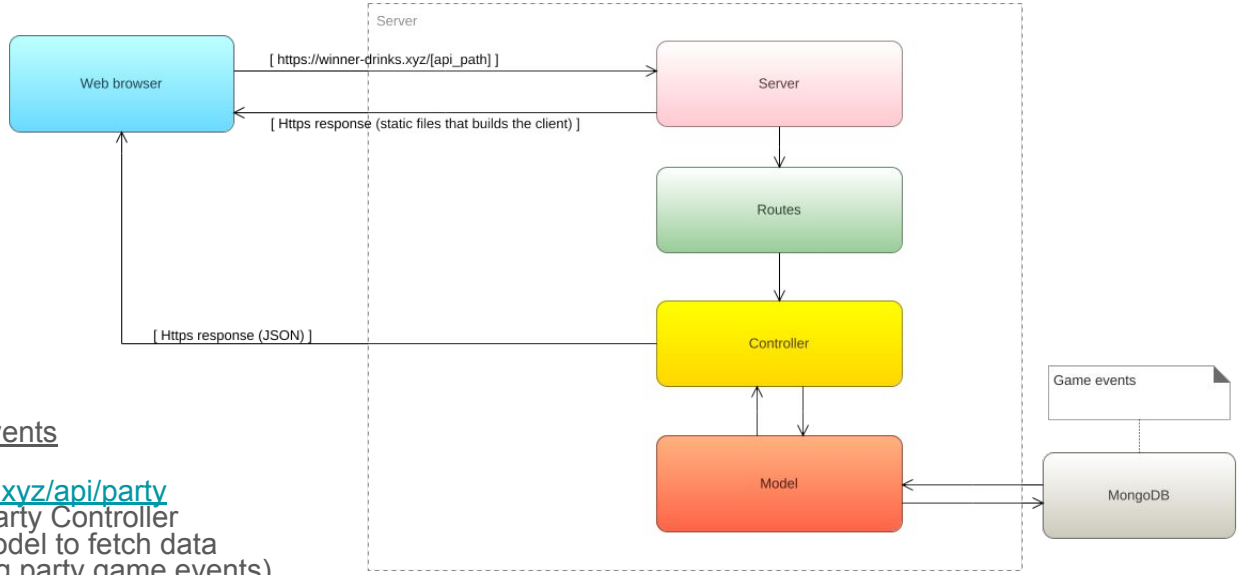
Server architecture and API

Server API, http GET requests

- /api/status
- /api/party
- /api/back-to-back
- /api/trivia

Only http requests when starting the app

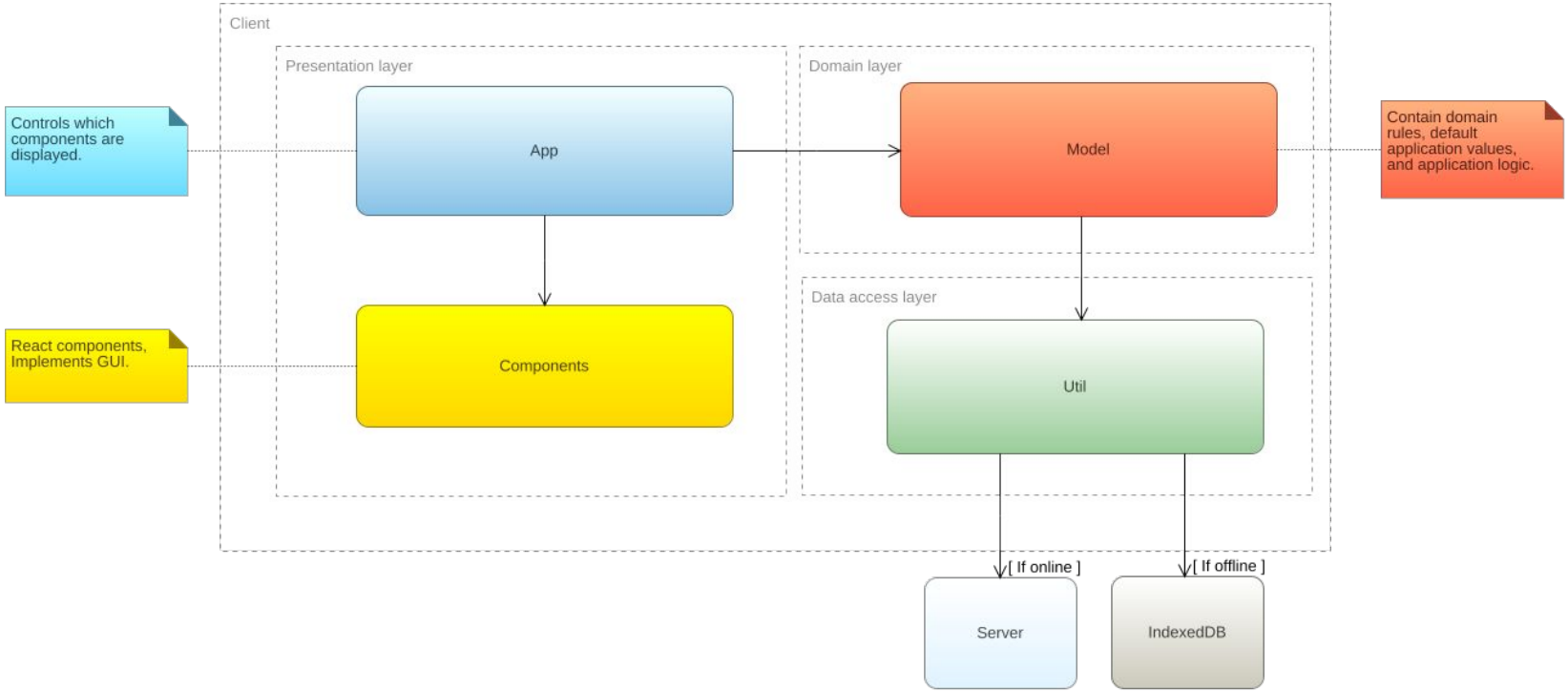
- Feels like a native application



E.g. client request data for party game events

- Client: https://www.winner-drinks.xyz/api/party
- Routes: Forward request to the Party Controller
- Party Controller: Use the Party Model to fetch data
- Party Controller: JSON(containing party game events)

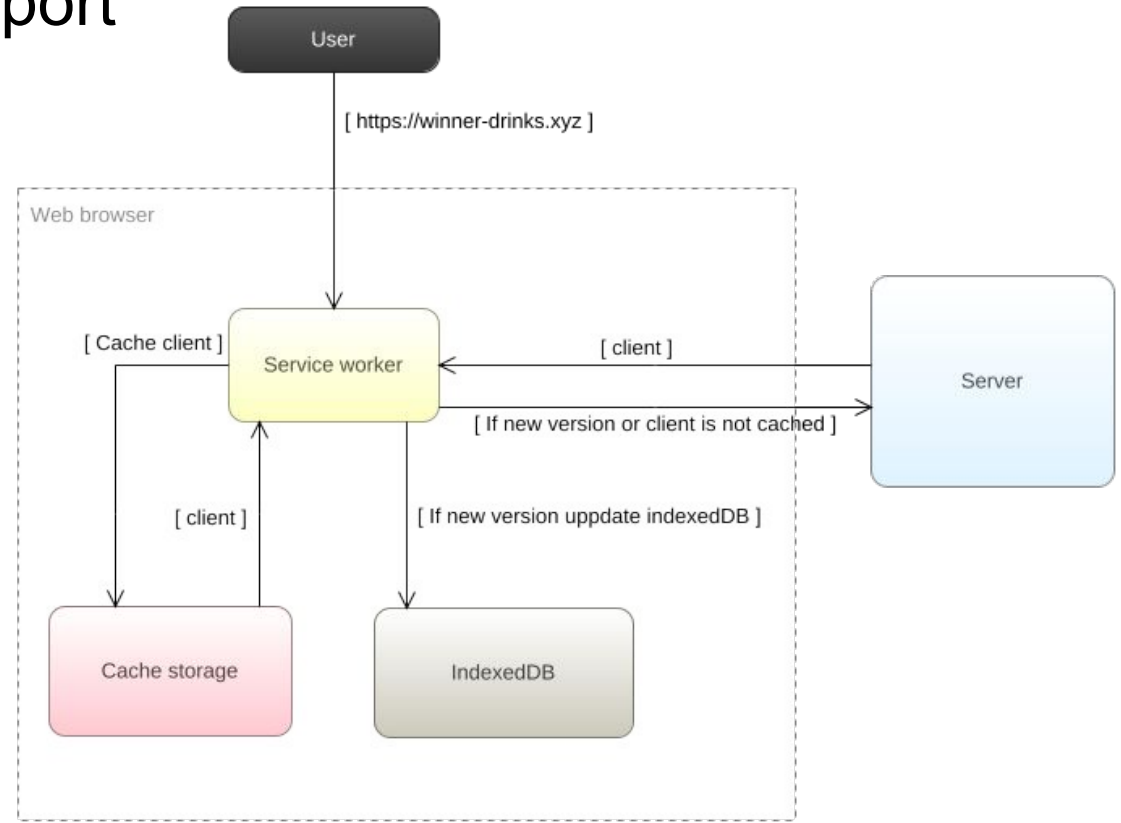
Client architecture, 3 tier architecture



Client and offline support

If service worker is supported

- ➔ Client register sw
- ➔ Act as a proxy server
- ➔ Cache the client
- ➔ Update IndexedDB



Demo