Observed Behaviour:

- User Experience
    - Backend hidden from end user
    - Easy access via the web
    - Intuitive GUI
        - New game button
        - Reset button
        - Virtual Keyboard
        - Color Coding
        - Aesthetic CSS styling
- Performance
    - Fast response times from local Java server
    - Smooth animations
    - Efficient word loading
- Error Handling
    - Errors on both the front end and back end are caught allowing for graceful degradation
    - Errors are printed to the console allowing for easy debugging

Architecture Overview:

- Frontend
    - apiService.js
        - Interacts with the backend server
            - Receives game state information
            - Sends user word guesses
    - app.js
        - The backbone of the frontend
            - Listens for user input
            - Handel's game logic
            - Connects to apiService.js, gamboard.js, and keyboard.js
    - gameboard.js
        - Stores the state of the game board
            - Manipulates the document to display the game board
            - Stores the current and previous guesses
    - keyboard.js
        - Manages the virtual keyboard
            - Takes virtual key inputs
            - Manages the status of keys (correct, present, absent)
- Backend
    - word.txt
        - A text file containing many 5 letter words
    - wordle folder
        - DTO folder

        - Contains various data transfer objects
            - GuessRequest
            - GuessResponse
            - ValidationResult
    - Exception folder
        - GameException.java
            - Maps error codes to their exceptions
    - Model folder
        - Has various classes which in combination are used to store the game state
    - Server folder
        - WordleHTTPServer
            - Sets up an HTTP server to handle requests from any connected clients

Method / Interaction Notes:

- API interactions
    - POST /api/game/new - creates new game with random word
    - GET /api/game/{gameId} - retrieves current game state
    - POST /api/game/{gameId} - submits guess with validation
    - CORS handling - enables cross-origin requests between frontend and backend
- Game logic flow
    - User submits words on the frontend web app
    - Guesses are sent to the WordleHTTPServer
    - Updated game state is sent back to the frontend
    - Repeat
- Frontend interactions
    - Keyboard Events - physical and on-screen keyboard support
    - Tile Management
    - Win/loss dialogs with replay options
    - Error Display - contextual messages for various failure types
- Implementation details
    - Simplified dev stack - removed Maven/Spring for simplicity and clarity
    - Defensive programming - null checks, exception handling, input validation
    - Responsive UI - keyboard interaction and visual feedback
    - User-friendly error handling
    - Input sanitization - case normalization and format validation
    - Response formatting - consistent JSON structure across all endpoints