**Beat the Computer!**

Welcome to the [**Beat the Computer!**](https://frontend-img-amd.onrender.com/) web app—an interactive, fast-paced experience built with modern web technologies. Below is a friendly rundown of the tech stack behind the project, how everything works together, and why each component was chosen.

**🌟Features & Highlights**

* **Microservices Architecture**: [Docker](https://www.docker.com/) efficiently scale frontend and backend separately.
* **Blazing Fast**: [Redis](https://redis.io/technology/data-structures/) caching keeps response times ultimately short.
* **Secure by Design**: [bcryptjs](https://www.npmjs.com/package/bcrypt) ensures secure handling and lightweight hashing of user passwords.

**Frontend**

Built using **React.js**, a powerful JavaScript library for creating interactive UIs effortlessly.

**Key Tools:**

* **React-Dom**: Essential for rendering React components to the DOM
* **React Router**: Seamless client-side navigation.
* **Tailwind CSS**: Easy, responsive styling with minimal hassle.

**Backend**

Powered by **Node.js** & **Express.js**, offering simplicity and high performance for API creation.

**Security:**

* **bcryptjs**: a popular NPM package that provides a C++-based implementation of the bcrypt algorithm. Encrypts passwords securely, ensuring user data stays protected. A good balance point between speed & security is when saltRounds = 10.

**Database:**

* **PostgreSQL** hosted on **Amazon RDS:**
  + Reliable, fully managed, and scalable database service (*more expensive than self-managed containerized database, but also less maintenance / easier scalability*).
  + Simplified database management with automatic backups.
  + Secure credentials management using *AWS Secrets Manager*.

**Caching & Session Management:**

* **Redis**: Instead of *MemoryStore*, provided by express-session and a basic in-memory store primarily intended for development and testing purposes, I chose Redis which efficiently handles real-time interactions, requests across multi-server instances, and can be easily deployed / connected on Render.
* **localStorage**: Utilized on the client side in conjunction with Redis and express-session, providing additional persistence and ensuring quick data access directly within the user's browser.

**🐳 Containerization**

* **Docker** ensures consistent application behavior across development, testing, and production.
* **Docker Compose** orchestrates frontend, backend, PostgreSQL, and Redis services seamlessly for effortless local development.

**Easy Deployment with Render**

Render is perfect for this application due to:

* Quick container deployments via **Docker Hub integration.**
* Immediate free-tier access without complicated setups or surprise bills (like other cloud services).
* Automatic HTTPS, scaling, and GitHub continuous deployments included.

**Try the Game!**

* [Beat the Computer!](https://frontend-img-amd.onrender.com/)

Feel free to explore and contribute :)

Happy coding and enjoy playing!