Complete Installation and Deployment Guide: Interactive Classroom System

This guide combines the initial installation process with deployment instructions, designed for setting up a new and clean environment on Firebase from scratch. The guide describes the manual process, and at the end presents scripts for automating the process.

Step 0: Prerequisites

Ensure the following tools are installed on your computer:

- **Node.js**: Including the `npm` package manager.
- **Firebase CLI**: If not installed, run in terminal:

  ```bash
  npm install -g firebase-tools
  ```

- **Accounts**: Google account and accounts for external API services (e.g., OpenAI, Anthropic, Gemini).

Step 1: Creating a Project and Environment in Firebase

This step is performed in the Firebase console.

1. Creating a New Project

- Go to the Firebase Console and create a new project with a unique name (e.g., `interactive-class-prod`).
- During setup, you'll be asked to choose a default location for resources (**Default GCP resource location**). **This step is critical**. Choose a location that will be identical for all services, for example **`europe-west1 (Belgium)`**.
- Ensure the **"Enable Google Analytics for this project"** option is checked.

2. Upgrading to Blaze Payment Plan

- After creating the project, go to billing settings (gear icon > `Usage and billing`).
- Upgrade the project to the **Blaze (Pay as you go)** plan. This is a mandatory step to use the Cloud Functions required by the system.

Step 2: Enabling Services and Setting Rules

Within the project you created:

1. Authentication

- In the side menu, go to `Authentication`.
- Go to the `Sign-in method` tab and enable the **Anonymous** authentication method.

2. Firestore Database

- Go to `Firestore Database` and click `Create database`.
- Choose to start in **Production mode**.
- Ensure the location selection is **identical** to what you chose in step 1 (e.g., `europe-west1`).
- After creating the database, go to the **Rules** tab, delete the existing content and paste the security rules from your project.

3. Secret Manager API

- Ensure this API is enabled for your project in Google Cloud Console. It's usually enabled automatically with Cloud Functions.

Step 3: Setting Up External API Keys

1. Create API keys for the services you'll use (Gemini, Claude, ChatGPT, etc.) through their official platforms.
2. Save the keys in a secure and accessible place for the next step.

Step 4: Setting Up the Local Project

1. Initialize Directory

- Create a new directory for the project, copy the source files to it, and open a terminal in it.

2. Link to Firebase Project

- Run the command:

```
bash

firebase init
```

- Choose **Use an existing project** and link it to the new project you created in Firebase.
- Select the following services: **Firestore**, **Functions**, and **Hosting**.
- Answer the initialization questions, but **avoid overwriting existing files** if asked (like `index.html` or `firestore.rules`).

3. Install Dependencies

- Navigate to the functions directory using `cd functions` and run `npm install` to install all required server-side dependencies.

Step 5: Local File Configuration

1. Setting Deployment Location (Functions)

- Open the `functions/index.js` file.
- Ensure the variable at the top of the file matches the location you chose. For example: `const DEPLOY_REGION = "europe-west1";`.

2. Creating Firebase Configuration File for Application (Frontend)

- **Recommended:** Run the following command in terminal (in the root directory) to automatically create the configuration file:

  ```
  bash

  firebase apps:sdkconfig WEB --out public/firebase-config.js
  ```

  This command generates the exact `firebase-config.js` file for your project.

3. Application Configuration (`config.json`)

- Ensure there's a file named `config.json` in the `public` directory.
- Ensure its structure is valid and the `studentAppUrl` field contains the complete address to the student application **in your project**.

Step 6: Setting Up Secrets (API Keys)

This is the secure way to manage your API keys. Run the following commands in terminal (from the root directory), and enter each key when prompted:

```
bash

firebase functions:secrets:set GEMINI_API_KEY
firebase functions:secrets:set CLAUDE_API_KEY
firebase functions:secrets:set OPENAI_API_KEY
```

Note: You can add more API keys as needed for your project.

Step 7: Deployment (Deploy)

1. (Optional) Build Step

If your project includes a build step (e.g., a script like `build.py` or `npm run build` that minifies files), run it now.

2. Final Deployment

To deploy the entire system (Hosting, Functions, Rules), run the command:

```
bash

firebase deploy
```

At the end of the process, you'll receive a new **Hosting URL**. This is the address of your new, live, and independent system.

Using Scripts for Automation

In addition to the manual process, the project includes scripts that allow you to perform some or all of the operations automatically.

Build Script (build.py)

This script is designed to prepare the Frontend files for deployment in a Production environment.

What does it do?

- Deletes the old BUILD directory if it exists and creates it anew.
- Goes through all files in the public directory.
- Minifies `.js` files (using Terser), `.css` and `.html` files (using minify).
- Copies other files (like `config.json` and `firebase-config.js`) as-is to the BUILD directory.

When to use? Run the script before deploying the project to upload a minified and efficient version to the server.

How to use?

1. Ensure you've installed the required dependencies shown at the top of the script (Terser, Minify).
2. Run the command:

```
bash
```

```
python build.py
```

Important: After running, update the `firebase.json` file so the Hosting directory is BUILD instead of public:

```
json
"hosting": {
  "public": "BUILD",
  "ignore": ["firebase.json", "**/.*", "**/node_modules/**"],
  "rewrites": [{"source": "**", "destination": "/index.html"}]
}
```

3. Run `firebase deploy`.

Automatic Installation and Deployment Script (deploy.py or install.py)

This script is a powerful tool that enables full automation of almost the entire manual process described in this guide. It's interactive and guides the user step by step.

What does it do?

- Checks system dependencies (node, npm, firebase).
- Guides the user in creating a new Firebase project through the terminal.
- Automatically enables required cloud services (`gcloud services enable`).
- Sets up local configuration files (`firebase.json`, `.firebaserc`, `config.json`).
- Runs dependency installation for functions (`npm install`).
- Securely guides API key entry and saves them as Secrets.
- Requests final approval before performing a general `firebase deploy`.

Simulation Mode: By default, the script runs in simulation mode. It only prints the commands it would run without actually executing them. This allows you to see what's about to happen before making real changes.

How to use?

For running in simulation mode (recommended for first time):

```
bash
python deploy.py
```

For running in "live" mode that performs all operations:

```
bash

python deploy.py --live
```

Note: The automatic installation script is an advanced tool. Use it carefully, especially in live mode, as it makes changes to your cloud project.

Troubleshooting

Common Issues

1. **Region Mismatch**: Ensure all Firebase services use the same region.
2. **Billing Issues**: Make sure your project is on the Blaze plan.
3. **API Key Problems**: Verify all API keys are correctly set using Firebase secrets.
4. **Build Errors**: Check that all dependencies are installed in the functions directory.

Support

For additional support, refer to the Firebase Documentation or check the project's issue tracker.

Security Considerations

- Always use Firebase secrets for API keys, never hardcode them.
- Regularly review and update your Firestore security rules.
- Monitor your Firebase usage to avoid unexpected charges.
- Keep your Firebase CLI and project dependencies updated.

Next Steps

After successful deployment:

1. Test all system functionality
2. Set up monitoring and logging
3. Configure backup procedures
4. Plan for scaling and maintenance

This completes the installation and deployment process for your Interactive Classroom System.