QUEST Deployment & User MANUAL

Team Codesauras

Table of Contents

1. Introduction

Welcome to the 2025S-Codesaurus Deployment Manual. 2025S-Codesaurus (also referred to as Quest) is an interactive web-based storytelling platform that blends compelling novel-like narratives with user choices, allowing readers to shape the story's progression. This full-stack application consists of a Node.js/Express backend (server API) and a React frontend (user interface). It integrates external services including the OpenAI API for dynamic content generation and AWS S3 for media storage, with MongoDB Atlas as its primary database.

This manual is intended for developers and system administrators who want to set up 2025S-Codesaurus for local development and deploy the application to production on Vercel. It provides step-by-step instructions for configuring the development environment, setting up all necessary services (database and third-party APIs), and deploying both the frontend and backend to Vercel's cloud platform. By following this guide, you will be able to run the application locally for testing and successfully deploy the full-stack application to a live environment using Vercel's default subdomains.

Scope of this Manual:

- System Requirements: Hardware/software requirements for development and production environments.
- Development Setup: Setting up the backend and frontend on a local machine for development and testing.
- Configuration: Setting up environment variables, database connections, and third-party service credentials (.env configuration for local use and Vercel environment variables for deployment).
- Deployment Procedure: Step-by-step instructions to deploy the frontend and backend to Vercel (using Vercel's default *.vercel.app subdomains).
- Verification & Testing: How to run the application and verify that all components (frontend, backend, database, APIs) work together in development and production.
- Troubleshooting: Common issues during setup or deployment and how to resolve them.

Whether you are running the application locally or deploying it to the cloud, this guide will ensure a smooth setup and deployment process for 2025S-Codesaurus.

2. System Requirements

To successfully develop and deploy 2025S-Codesaurus, your system and tools should meet the following minimum requirements. This includes hardware capabilities for running the app locally, software installations for development, and accounts or services needed for production deployment.

2.1 Hardware Requirements

The hardware requirements for running the application (especially during development) are modest, but using a machine that meets the recommended specifications will ensure a smoother experience:

Component	Minimum	Recommended
Processor	Dual-core 2.0 GHz (e.g. Intel i3)	Quad-core 2.5 GHz (e.g. Intel i5/i7 or equivalent)
Memory (RAM)	4 GB	8 GB or more
Storage	10 GB free (for source code, Node modules, etc.)	20 GB free (to accommodate additional logs or data)
Internet	Broadband connection	Stable high-speed internet (for API calls and package downloads)

Note: A stable internet connection is required during development and production use, as the application depends on external services (OpenAI API, MongoDB Atlas, AWS S3). In production, the Vercel platform will handle hardware needs, but internet connectivity is essential for end-users and service integrations.

2.2 Software Requirements

Ensure the following software is installed on your development machine:

- Operating System: Windows 10/11, macOS, or Linux. (The project is OS-agnostic; use whichever OS you are comfortable with.)
- Node.js and NPM: Node.js (JavaScript runtime) version 16.x or 18.x LTS or higher, along with its package manager NPM (which comes with Node). This is required for both the backend and frontend development. Verify installation by running *node -v* and *npm -v* in a terminal.
- Git: Git for version control (to clone the repository). Alternatively, you can download the project as a ZIP from GitHub, but Git is recommended.
- Code Editor/IDE: (Optional but recommended) An IDE or text editor for code editing. For example, Visual Studio Code is recommended for both frontend and backend development. Other options: WebStorm, Atom, or any editor of your choice.
- Web Browser: A modern web browser for testing the frontend. Google Chrome, Mozilla Firefox, or Microsoft Edge (latest versions) are recommended for development and debugging.

2.3 Development Environment

In a development environment (your local machine), you will use the above tools to run the application locally:

- Node.js & NPM: Used to install dependencies and run the local development servers for both backend and frontend.
- Local Environment Variables: A *.env* file will be used to store configuration like database connection strings and API keys. (Details on setup in a later section.)
- MongoDB Atlas Access: You do not need a local MongoDB server; the app uses MongoDB Atlas (cloud database). However, ensure you have the connection URI and credentials ready.
- Optional Tools:
    - Postman or cURL: for testing backend API endpoints directly during development.
    - nodemon: for automatically restarting the Node.js server on code changes (if not already configured in the project).
    - Vercel CLI: for interacting with Vercel from the terminal (optional, only needed if you prefer command-line deployment or pulling environment configs).

2.4 Production Environment

For production, 2025S-Codesaurus will be deployed on Vercel, a cloud platform for hosting web applications. You do not need to manage physical servers, but you will need to prepare the following:

- Vercel Account: Sign up for a free account on Vercel (using GitHub for login is convenient). No credit card is required for the free tier.
- GitHub Repository Access: The source code will be deployed via the GitHub repository. Ensure you can connect the GitHub repo (*htmw/2025S-Codesaurus*) to Vercel. If it's a private repository, you'll need appropriate access rights.
- Environment Variables on Vercel: You will configure the same keys from your *.env* in Vercel's dashboard for the production deployment (to connect to MongoDB, OpenAI, AWS, etc.). Vercel allows setting encrypted environment variables through the Project Settings.
- MongoDB Atlas Cluster: A cloud database instance accessible by the Vercel backend. (Ensure network permissions in Atlas allow Vercel to connect, e.g., IP whitelisting or 0.0.0.0/0 for testing.)
- OpenAI & AWS Services: Active service accounts (OpenAI API enabled, AWS account with S3 bucket) since production will make live calls to these services.

Vercel will provide default subdomains (e.g., *your-project-name.vercel.app*) for each deployed project. We will use these default HTTPS URLs for accessing the app in production. Vercel automatically handles scaling, SSL certificates, and server infrastructure for you, so you don't need to provision servers or containers manually. Just ensure your third-party services (database, API, storage) are properly configured, as detailed in upcoming sections.

2.5 Optional Tools

While not strictly required, the following tools and services can assist in development and deployment:

- Postman – to test and debug API endpoints of the backend by simulating requests.
- Vercel CLI – a command-line tool to deploy projects and manage environment variables on Vercel from your terminal. This can be useful for deploying the backend or pulling environment settings locally.
- MongoDB Atlas UI / Compass – to view and manage the data in your MongoDB database (helpful for verifying that data is being written correctly).
- AWS Management Console – to inspect the S3 bucket (to verify that images or files are uploaded correctly).
- Logging Tools – Vercel provides serverless function logs in its dashboard. Familiarize yourself with Vercel's logging for debugging production issues.

Having these tools can streamline troubleshooting and give more visibility into the application's behavior both locally and in production.

3. Installation and Setup (Development Environment)

This section covers the step-by-step setup for running the 2025S-Codesaurus application on your local machine (development environment). You will be setting up both the backend (server API) and the frontend (React application). By the end of this section, you should have the backend server running locally and the frontend accessible in your browser, communicating with the backend.

3.1 Setting Up the Backend (Development)

Follow these steps to set up and run the backend API locally:

Clone the Repository: If you haven't already, clone the project repository to your local machine using Git. In a terminal, run:

git clone https://github.com/htmw/2025S-Codesaurus.git

This will create a directory 2025S-Codesaurus containing the project files. (Alternatively, download the repository ZIP and extract it.)

Install Backend Dependencies: Navigate into the backend folder in the repository. The backend code is located in the Backend directory. In terminal:

cd 2025S-Codesaurus/Backend

npm install

This will install all Node.js dependencies for the backend as specified in package.json. Ensure the installation completes without errors and that a node_modules folder is created.

Configure Backend Environment Variables: The backend relies on certain configuration values (database URI, API keys, etc.) which should be set in a *.env* file in the Backend directory. A sample environment file (*.env*) has been provided with the project. Open the Backend/*.env* file (or create a new .env file if it's not present) and ensure it contains the required keys (see Section 8: Environment Variables for a detailed list). At minimum, set:

MONGO_URI – MongoDB Atlas connection string

OPENAI_API_KEY – your OpenAI API key

AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY – AWS credentials for S3

AWS_REGION – AWS region of your S3 bucket

S3_BUCKET_NAME – name of your S3 bucket

PORT – port for local server (default 8081)

Save the .env file. (Do not commit this file to version control as it contains secrets.)

Start the Backend Server: Launch the backend API server on your local machine. Usually, you can do this by running an npm script or directly using Node:

npm start

If *npm start* is not defined for the backend, use the main file directly. For example, if the main server file is *index.js* or *app.js*, run: *node index.js*. The server should start, and you should see console output indicating that it's running (e.g. "Server listening on port 8081"). By default, the backend will listen on port 8081 (as defined by the *PORT* in .env).

Verify Backend is Running: Open a browser or use a tool like curl/Postman to test a simple endpoint. For example, the backend may have a health check endpoint or root route. Try accessing *http://localhost:8081/* (or an appropriate endpoint if documented) in your browser. If configured, it might return a message (e.g., "API is running"). If the project has no open root route, you can still confirm by checking the terminal output – it should show no fatal errors. Keep the backend server running for the next steps.

(At this stage, the backend API is up and waiting for requests. It should be connected to the database (MongoDB Atlas) if the *MONGO_URI* is correct, and ready to use the OpenAI and AWS services when requests arrive. Now we will set up the frontend.)

3.2 Setting Up the Frontend (Development)

Now set up and run the React frontend:

Install Frontend Dependencies: Open a new terminal window (keeping the backend running in the other) and navigate to the frontend directory of the project.

cd 2025S-Codesaurus/frontend

npm install

This installs all the React app's dependencies as listed in its *package.json* (such as React libraries, UI frameworks, etc.). Wait for all packages to finish installing. If there are no errors, you'll have a *node_modules* directory in the *frontend* folder.

Configure Frontend Environment (if needed): Typically, the frontend of 2025S-Codesaurus is a React application that communicates with the backend via API calls. In development mode, if the project was set up with a proxy or CORS, you might not need to manually configure the API URL. Check the frontend configuration:

If using Create React App, there might be a proxy defined in *package.json* (e.g., "proxy": "*http://localhost:8081*"). This means during development, API calls from the React app to unknown paths will be forwarded to the backend on port 8081, simplifying local development.

If not using a proxy, you may need to set a environment variable for the frontend with the backend's URL (e.g., a React app would use *REACT_APP_API_URL*). In the development *.env* for the frontend (e.g., *frontend/.env.local*), you could set *REACT_APP_API_URL=http://localhost:8081*. This step is optional and only needed if the code expects such a variable. In many cases, the local dev may already assume *localhost:8081*. (We will cover production configuration in the deployment section.)

Start the Frontend Development Server: Run the React app in development mode with:

npm start

This should launch the development server (often using react-scripts if Create React App, or Next.js dev server if it was a Next app). The terminal will show compilation progress, and once compiled, it will usually open a browser window automatically. By default, the React app runs at http://localhost:3000.

Access the Frontend UI: If the development server didn't open automatically, open your web browser and navigate to *http://localhost:3000*. You should see the 2025S-Codesaurus application's frontend interface (e.g., a homepage or login screen for the interactive novel platform). Confirm that the page loads without errors. The frontend is now running and should be trying to communicate with the backend API.

Verify Frontend-Backend Integration: To ensure the two parts are connected:

- Perform an action in the frontend that triggers an API call to the backend. For example, if the app has a login, try to register/login; if it has a "start story" button, click it.
- Check that the backend terminal logs show some activity (like incoming requests or responses). Also check the browser dev console (F12) Network tab for the API

requests. They should be going to *localhost:8081* endpoints and returning appropriate responses (200 OK or expected data).

- If requests from the frontend are failing, it could be a CORS issue or incorrect API base URL. In development, if both are on localhost but different ports, ensure the backend has CORS enabled (or use the proxy method). If needed, you can enable CORS in the backend by using a library like cors in Express to allow *localhost:3000* or all origins during development.

Local Development Ready: At this point, you have:

- Backend running on http://localhost:8081 (providing API endpoints).
- Frontend running on http://localhost:3000 (providing the user interface).
- Both should be working together. Keep these running as you develop. You can make code changes and refresh the browser to see updates (frontend will hot-reload for UI changes; if you change backend code, you may need to restart it unless using a live reload tool).

3.3 Testing the Local Setup

Before moving on to deployment, test the main features locally:

- Basic Navigation: Click through the frontend UI to ensure all pages/components load (no missing dependencies or broken React routes).
- Database Operations: Try an operation that should store or fetch data from MongoDB Atlas (e.g., creating a user profile, or saving a choice in the story). Then verify (possibly via MongoDB Atlas UI) that the data was created in the database. If the application has a feature to view or list data, use that to confirm data persistence.
- OpenAI Integration: If the app uses OpenAI (for generating story content or NPC dialogue), test that functionality. For example, start a new story or perform the action that triggers an AI-generated text. The response might take a moment (due to API call). Ensure you get a plausible result and no error message. If there's an error related to OpenAI, check that your *OPENAI_API_KEY* is correct and that you have an active internet connection.
- AWS S3 Integration: If the app involves uploading or retrieving images (such as character images or user-uploaded content), test an upload. For instance, if there is an avatar upload or any image generation feature, perform it. Then check the AWS S3 bucket (via AWS Console) to see if the image file appears in the *quest-npc-images* bucket. If not, check backend logs for errors (it might be an AWS credentials or permission issue).
- Logging and Errors: Watch the terminal output for both backend and frontend. Address any errors that appear. Common issues might include missing environment variables (if something wasn't set in .env, the app might warn or crash), CORS errors (if frontend couldn't talk to backend), or wrong API endpoints (404s if the front is calling a path that backend doesn't have).

By thoroughly testing the integration locally, you can be confident that your configuration is correct before proceeding to deploy the application to Vercel. Once everything looks good, stop the development servers (Ctrl+C in each terminal) and prepare for deployment.

4. Database Configuration (MongoDB Atlas)

2025S-Codesaurus uses MongoDB Atlas as the database to store application data (such as user accounts, story choices, etc.). In the development setup, we referenced MONGO_URI in the environment variables. In this section, we detail how to configure MongoDB Atlas for the application. Using the Provided MongoDB URI: The project's *.env* (provided) contains a *MONGO_URI* in the following format:

MONGO_URI=mongodb+srv://<username>:<password>@<cluster>.mongodb.net/<databaseName>

For example, it might look like:

mongodb+srv://QuestAll:AllQuest123@quest.b19sh.mongodb.net/Quest

This URI includes the database user credentials and the target database name (in this example, database name is "Quest"). If you have been given this URI and the credentials, you can use it directly. Ensure that the *MONGO_URI* is correctly placed in the backend's .env file.

Setting Up Your Own MongoDB Atlas (if needed): If you prefer to use your own MongoDB Atlas cluster (or if the provided one is not accessible), follow these steps:

- Create a MongoDB Atlas Account: Sign up at MongoDB Atlas and create a new project. Create a new free cluster (M0 tier) if one is not already available.
- Create a Database User: In your cluster's Security > Database Access, add a new database user. Give it a username and password. Copy these credentials for use in the connection string.
- Network Access: Under Security > Network Access, ensure the IP address of your environment is allowed. For development, you can allow access from anywhere (0.0.0.0/0) so your local machine and Vercel (production) can connect. (Be cautious: allowing all IPs is convenient for development but less secure for production databases.)
- Create the Database and Collections: The connection string includes a database name (for example, "Quest"). This database will be created automatically when the backend first writes data to it. You may create it in advance through the Atlas UI. Likewise, you can set up initial collections if you know them, or let the application create them on first use.
- Update MONGO_URI: Construct your connection string using the format above. In Atlas, you can find the connection URI in the cluster connect settings (choose "Connect your application"). It will be something like:

mongodb+srv://<username>:<password>@<your-cluster url>.mongodb.net/<YourDB>?retryWrites=true&w=majority

Replace *<username>, <password>,* and *<YourDB>* with your values. Add this to the *.env* file as *MONGO_URI*. Example:

MONGO_URI=mongodb+srv://admin:MyPass123@cluster0.abcde.mongodb.net/codesaurus

(If you use special characters in username/password, ensure they are URL-encoded.)

Test the Connection: With the backend running locally, you should see in the logs if the connection to MongoDB is successful. If there is a connection error, check that:

- The URI is correct (no typos in cluster name, user, password).
- The database user has permissions (Atlas users by default can read/write any database if set to admin roles).
- Your IP is allowed in Atlas security settings.
- No firewall is blocking the connection from your machine.

In production (Vercel), the backend will use the same *MONGO_URI*. You must ensure the Atlas cluster is accessible from Vercel. Typically, allowing access from anywhere in Atlas or adding Vercel's IP ranges (which can vary) is required. If you used *0.0.0.0/0* during dev, it will also cover Vercel. Just remember this setting for a production environment, and restrict if necessary once deployed (for better security, you could later whitelist specific IPs or use Vercel's Atlas integration if available). MongoDB Collections and Data: The specific collections (tables) used by the application will depend on the project's design (users, storyline, choices, etc.). Check the project documentation or code for schema details. The application should create and manage these collections automatically via the backend code when it runs. No additional setup (like migrations) is typically needed unless specified. By properly configuring the *MONGO_URI* and ensuring connectivity, the database part of the application should work seamlessly in both development and production. All data persistence in 2025S-Codesaurus will rely on this connection, so double-check this configuration if you encounter any data-related issues.

5. Third-Party Services Configuration

Apart from the database, 2025S-Codesaurus integrates with two main third-party services: OpenAI API and AWS S3. These services require separate setup and credentials. In your environment variables, you have placeholders for their keys (like OPENAI_API_KEY, AWS_ACCESS_KEY_ID, etc.). This section explains how to set up and obtain those credentials and any necessary configuration for these services.

5.1 OpenAI API Key Setup

The application uses OpenAI's API (likely to generate text for the interactive story or related AI features). To use the OpenAI API, you need an API key:

- Obtain an OpenAI API Key: Sign up at the OpenAI developer platform and log in to your account. Navigate to the API Keys section of your account (often under user settings or a dedicated page). If you don't have a key yet, create a new secret key. Copy the API key – it usually starts with sk-... and is a long string. (For example, the provided .env shows a key beginning with sk-proj-... which is a format of an OpenAI key.)
- Set the Key in .env: Open your backend's .env file and find OPENAI_API_KEY. Replace its value with your copied key (or ensure the provided key is correct if one was given for your use). It should look like:
- OPENAI_API_KEY=sk-XXXXXXXXXXXXXXXXXXXXXXXXXXXX (with your actual key in place of the Xs).
- Security Note: Do not expose this key in frontend code or commit it to any public repo. The OpenAI API key is a secret credential. According to OpenAI's best practices, "The OpenAI API uses API keys for authentication. Visit your API Keys page to retrieve the API key you'll use in your requests. Remember that your API key is a secret! Do not share it with others or expose it in any client-side code.". In our setup, we keep it on the backend (so it's secure on the server side and not visible to users).

No further configuration is typically needed for OpenAI API usage, aside from ensuring that the backend code knows which model or endpoint to call. The API key gives access to OpenAI's services (within usage limits).

Testing OpenAI Integration: After setting the key, when running the app (locally or on Vercel), test a feature that uses OpenAI. If there's an issue (e.g., you get errors or no response), possible causes include:

- The API key is invalid or not properly set (double-check the value and that the backend process sees it via *process.env.OPENAI_API_KEY*).
- You might have hit a usage limit or need to enable billing for higher usage (OpenAI's free tier has certain limits).
- The code might be expecting a specific environment variable name; using the provided name should be correct.

If needed, you can log or output a message from the backend on startup to confirm it loaded the OPENAI_API_KEY (just don't print the key itself; simply indicate that it's present or not null).

5.2 AWS S3 Bucket and Credentials

The application uses AWS S3 for storing media assets, such as images. In our case, it appears to store "quest NPC images" in a bucket (as suggested by the bucket name quest-npc-images). To set this up:

- AWS Account and IAM User: You will need access to an AWS account. If you don't have one, create an AWS free-tier account. Then, do the following in AWS:
    - Go to the AWS IAM (Identity and Access Management) service. Create a new IAM User dedicated for this application (e.g., username codesaurus-s3-user).
    - Assign permissions to this user to access S3. The simplest way is to attach the managed policy AmazonS3FullAccess for testing/development purposes. For more restrictive access, create a custom policy that only grants access to the specific bucket (quest-npc-images) with appropriate actions (GetObject, PutObject, etc.).
    - After creating the user, under its Security Credentials, generate a new Access Key. This will provide you with an Access Key ID and a Secret Access Key (you can only copy the secret once at creation time).
    - Create S3 Bucket: Go to the AWS S3 service. Create a new bucket named quest-npc-images (if you want to use a different name, note that you must update the S3_BUCKET_NAME in the environment variables to match). Choose a region (for example, us-east-2 as provided in .env). You can keep it public or private depending on needs; likely this bucket might store images that are accessed through the application, so setting it public-read could allow the frontend to fetch images directly if needed. For now, create the bucket with default settings and note the region.
    - Set AWS Credentials in .env: Take the Access Key ID and Secret from the IAM user you created and put them in the backend .env:

      AWS_ACCESS_KEY_ID=YOURACCESSKEYID

      AWS_SECRET_ACCESS_KEY=YourSecretKeyHere+ABC123...

      AWS_REGION=your-bucket-region (e.g., us-east-2)

      S3_BUCKET_NAME=quest-npc-images

      Replace with your actual values. The region should match where you created the bucket. The bucket name should match exactly the name in AWS.

    - Verify AWS Credentials: The backend will use these credentials to initialize an AWS SDK client for S3. It's important the IAM user has correct permissions. Essentially, the IAM user must have rights to at least read and write to the specified bucket. In summary, "you need to create an access key/secret key pair which is associated with an IAM user. You need to ensure the IAM user has appropriate permissions to access S3.". Without correct permissions, you might get errors when the app tries to upload or fetch files.
    - Testing S3 Integration: With the app running (locally), perform an operation that involves S3. For example, if there's a feature to upload an image or generate an image, do that. Then check:

- The operation's result in the app (was it successful? any error messages?).
- The S3 bucket via AWS Console to see if a new object (file) appeared.
- The backend logs for any error messages related to AWS (e.g., permissions or missing configuration errors).

If you see permission errors, revisit the IAM user's policy. If you see connectivity or credential errors, ensure the keys are correct in .env (no extra quotes, no missing characters) and that *AWS_REGION* is correct.

The AWS S3 integration allows the application to offload file storage to the cloud, which is ideal for a scalable deployment on Vercel (since Vercel's serverless functions are ephemeral and should not store files locally). Once properly set up, uploaded files will persist in S3 and can be retrieved when needed by the app.

With MongoDB, OpenAI, and AWS configured, the next step is to ensure all these configuration values are correctly fed into your application via environment variables. We will now summarize the environment variables and how to manage them, especially when deploying to Vercel.

6. Environment Variables and Configuration

Throughout the previous sections, we identified several environment variables critical to running 2025S-Codesaurus. During local development, these values are stored in the .env file in the backend directory. In production (Vercel), they must be configured in the Vercel project settings. This section will list all necessary environment variables and provide instructions for configuring them in both development and production.

6.1 Summary of Environment Variables

Below is a table of the environment variables used in 2025S-Codesaurus, along with their descriptions and example values:

Environment Variable	Description	Example Value
MONGO_URI	MongoDB Atlas connection string (including username, password, cluster URL, and database name). Allows the backend to connect to the MongoDB database.	*mongodb+srv://QuestAll:AllQuest123@quest.b19sh.mongodb.net/Quest*
PORT	Port number on which the backend server runs (development use). Not typically needed in production (Vercel will assign a port) but used locally.	*8081*
OPENAI_API_KEY	Secret API Key for OpenAI. Used by the backend to authenticate with OpenAI API for generating content. Keep this secret (do not expose on frontend).	*sk-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*
AWS_ACCESS_KEY_ID	AWS IAM Access Key ID for the user with S3 access. Provided by AWS when you create an access key for an IAM user.	*AKIAIOSFODNN7EXAMPLE*
AWS_SECRET_ACCESS_KEY	AWS IAM Secret Access Key for S3. Provided alongside the Access Key ID. Keep this secret.	*wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY*
AWS_REGION	AWS region where your S3 bucket is hosted. Ensures the AWS SDK targets the correct region.	*us-east-2*
S3_BUCKET_NAME	The name of the AWS S3 bucket used to store images or files for the app.	*quest-npc-images*

(The above example values are illustrative. Replace them with the actual values corresponding to your setup. The provided .env file in the project contains placeholders or example credentials which you should update with real credentials as described in sections 4 and 5.)

If the frontend also requires configuration (for example, a variable for the backend API URL), you may also have a variable like REACT_APP_API_URL in a frontend .env or build config. That would be used only on the frontend side. (In production, you could set REACT_APP_API_URL in Vercel to tell the built frontend where to find the backend API. See section 9.3 on connecting frontend and backend.)

6.2 Using .env in Local Development

For local development, ensure that the *.env* file resides in the *Backend* directory (or wherever the backend code expects it). The Node.js backend will typically load this file (often via a library like dotenv) on startup, making the variables available via *process.env*. Tips:

- Do not commit *.env* to Git. It should be listed in *.gitignore* (the provided project likely already ignores it). This keeps secrets out of version control.

- If you have multiple environment files (some projects use *.env.development*, *.env.production*), ensure you are editing the correct one. In our case, one .env file for development is sufficient.
- If you make changes to *.env* while the local server is running, you'll need to restart the backend to pick up the new values.
- Double-check that every variable in the table above is present and filled in your .env before running the app.

6.3 Configuring Environment Variables on Vercel

When you deploy to Vercel, you cannot use the local .env directly on the cloud for security reasons. Instead, you will configure these values in your Vercel project settings:

1. Open Vercel Dashboard: Go to your project's settings on Vercel (you will have one project for the backend API, and possibly one for the frontend, if deploying separately).
2. Environment Variables Section: In the project settings, find the Environment Variables tab. Here you can add the key-value pairs for each environment variable. Vercel lets you specify values for Development, Preview, and Production environments. Since we are focusing on production deployment, you can add the variables under Production (and optionally under Preview for staging deployments).
3. Add Each Variable: For each variable (MONGO_URI, OPENAI_API_KEY, etc.), click "Add New". Enter the Name (e.g., MONGO_URI) and the Value (the actual connection string or key). For the environment, choose Production (and repeat for Preview if you intend to use preview deployments). Do not expose these values to the front end (leave the checkbox "Encrypt" as is, which is default). They will be available to the backend Node environment. Vercel encrypts these values at rest and ensures they are injected during the build and runtime.
4. Frontend Variables: If the frontend needs a variable (like REACT_APP_API_URL or any config), add it in the frontend project's settings similarly. Remember that for React (Create React App) or other frontend frameworks, any variable that needs to be accessible in browser JavaScript must be prefixed (e.g., REACT_APP_ for CRA, or NEXT_PUBLIC_ for Next.js). In our case, if we use REACT_APP_API_URL on the frontend, set that in Vercel for the frontend project.
5. Redeploy if Needed: Important: After adding environment variables on Vercel, you will need to redeploy (or trigger a rebuild) of the project for them to take effect, as environment variables are loaded at build and runtime. If you add a variable but do not redeploy, the running app won't see it.

By configuring these in Vercel, you achieve the same effect as the .env locally. For example, process.env.MONGO_URI in your Node code will now have the Atlas URL value when running on Vercel. This separation of config from code is key to managing different environments safely.

Verification on Vercel: After deployment, you can verify that environment variables are working:

- In the Vercel dashboard, check the Build Logs to see if any errors occur connecting to database or other services (which might indicate an env var not set).
- You can also use Vercel's CLI command *vercel env pull* to download the variables to a local .env file for verification or debugging, if needed.
- If an environment variable is not taking effect, ensure you added it to the correct project and environment. Printing out *process.env* values in the logs (for debugging) can help, but be cautious not to log secrets. Vercel's documentation suggests logging or echoing the env var in a build step as a troubleshooting measure.

With environment variables properly configured on Vercel, your deployed application will have access to the same keys and connection strings as your local version did via .env. Next, we will cover how to deploy the frontend and backend to Vercel, tying together all these configurations.

7. Running the Application (Local Verification)

(If you have already tested the integration in Section 3.3, you may skip to Section 8 for deployment. Otherwise, this section reiterates the steps to verify the complete application before deploying.) Before deploying to production, it's important to run the entire application locally and verify that all components work together. This ensures that any configuration issues can be caught early. Here's a quick checklist for verifying the running application in the development environment:

- Start Backend and Frontend: Make sure the backend API server is running on port 8081 and the frontend dev server on port 3000, as described earlier. Both consoles should be free of errors.
- Access the App in Browser: Navigate to http://localhost:3000 in your web browser. You should see the Codesaurus application UI. Perform some basic interactions.
- User Authentication (if applicable): If the app has a login/signup, try creating a test account or use any seed/test credentials provided. Ensure that account creation is successful (the backend should write to MongoDB Atlas). Check MongoDB Atlas to see the new user document in the appropriate collection.
- Start a Quest (Story): Go through the core functionality — for example, begin a new interactive story. As you make choices in the story:
    - The frontend should update the narrative based on your choices.
    - The backend might be logging each request that fetches the next part of the story. No errors should be logged.
    - If the story content is AI-generated (OpenAI), ensure responses come through within a reasonable time. If it's very slow or errors out, check the OpenAI configuration.

- Image Uploads/Displays: If there is a feature to display images (like NPC avatars or story illustrations), check that they appear. If the images are fetched from S3, the URLs might be pointing to the S3 bucket. Ensure those are loading (check browser console for any 403 errors from S3 — if so, you might need to adjust bucket permissions to allow public read).
- Data Persistence: Quit the application (stop backend, stop frontend). Then restart them and check if data persists:
    - For example, log in with the account you created earlier to make sure it was saved in the DB.
    - Continue a story or see if previous choices were recorded (if such a feature exists).
    - Essentially, ensure the state that should be stored in the database or S3 is actually stored and can be retrieved on a fresh start.
- Integration between Components: Observe that the frontend and backend are communicating properly:
    - No CORS errors in the browser console (in development, either the proxy or CORS config should handle it).
    - The correct API endpoints are being hit (use browser dev tools Network tab to inspect calls; they should be going to localhost:8081 for API data).
    - The backend is sending appropriate responses (HTTP 200 for success, etc., which you can see in Network tab or backend logs).

By completing this testing, you have confidence that the app works in a controlled environment. Now you are ready to deploy it to Vercel.

(If any issues were encountered, resolve them now — check earlier sections for configuration or consult project-specific documentation. Common issues like backend not running (check Node version, dependencies), frontend failing to compile (check Node/NPM versions), or inability to connect to database (check MONGO_URI and network settings) should be addressed before proceeding.)

8. Deploying to Vercel (Production)

Deployment to Vercel will involve hosting the frontend and backend as two separate projects on Vercel. We will use Vercel's default subdomains for each (for example, your-frontend-app.vercel.app and your-backend-api.vercel.app). The frontend (React app) will be deployed as a static site (with client-side rendering), and the backend will be deployed as a serverless Node.js function (via Vercel's Node runtime). Below are the steps to deploy both components:

8.1 Prepare the Repository for Vercel

Vercel supports monorepos (multiple projects in one repository) by allowing you to specify a subdirectory as the root for each Vercel project. In our case, we have one repository

containing two subfolders *(/frontend and /Backend*). We will set up two Vercel projects from this single repo:

- Frontend Project: Root directory will be frontend.
- Backend Project: Root directory will be Backend.

Ensure that your repository is pushed to GitHub and that you have access to connect it to Vercel. If you have any build or configuration files specifically needed for Vercel (for example, a vercel.json in the backend to define the entry point), have those ready (we'll cover the vercel.json for the backend below).

8.2 Deploying the Frontend on Vercel

- Create New Project on Vercel: Log in to Vercel and click "Add New... > Project". Choose the GitHub repository htmw/2025S-Codesaurus. Vercel will then prompt you to configure the project.
- Set Root Directory: Since this is the frontend project, click "Edit" next to Root Directory (when prompted) and select the frontend folder. This tells Vercel to only deploy the code within */frontend*.
- Build and Output Settings: Vercel should auto-detect the framework. For a React app (Create React App), it will likely detect it and set the default Build Command as *npm run build* and Output Directory as *build/*. Verify these settings:
    - Framework Preset: Create React App (if applicable, or set None if it's a plain React).
    - Build Command: *npm run build*
    - Install Command: (leave as default *npm install*, or use Yarn if your repo used Yarn).
    - Output Directory: *build*
- Environment Variables: Configure any environment variables needed for the frontend (if any). For example, if the frontend code expects a *REACT_APP_API_URL* for the backend's URL in production, set it here under "Environment Variables" for the project. (You can also add this after initial deployment in the project Settings.)
- Deploy: Click "Deploy". Vercel will start the build process: it will install dependencies and run *npm run build* to create a production build of the React app. If everything is configured correctly, the build will succeed and Vercel will deploy the static files.
- Preview the Frontend Deployment: Once deployed, Vercel will assign a default domain like *your-project-name.vercel.app*. Visit this URL in your browser. You should see the frontend's initial page. Note that at this stage, the frontend might not fully work because the backend isn't deployed yet (or the API URL might not be configured). That's okay—we will deploy the backend next.
- Post-Deploy Config (if needed): After the first deployment, you can go to the Vercel dashboard for this frontend project and double-check:
    - That the environment variable for API URL is set to the backend's URL (we will know the backend's URL after deploying it). You can set it now if you

anticipate the name (e.g., *https://codesaurus-backend.vercel.app* if you plan to name the backend project "codesaurus-backend"). Or you can set it after backend deployment.

- Ensure the frontend is using HTTPS when calling the backend. The default Vercel domain uses HTTPS. If your frontend code was using *http://localhost:8081* for dev, make sure in production it will use *https://<backend-domain>*. Usually, setting REACT_APP_API_URL to an *https://...vercel.app* address is the solution.

8.3 Deploying the Backend on Vercel

Next, deploy the backend as a separate Vercel project:

- Create Backend Project: In Vercel, add another new project from the same repository. (Vercel will warn that the repo is already linked, but it allows multiple projects from one repo.) When configuring, set the Root Directory to *Backend* for this project.
- Set Project Name: Give it a clear name like *codesaurus-backend* (this will influence the default domain, e.g., *codesaurus-backend.vercel.app*).
- Configure Build Settings: The backend isn't a static site; it's a Node.js application. Vercel can deploy Node apps using serverless functions. We need to instruct Vercel how to start the backend. There are two approaches:
  - Using *vercel.json*: The repository should have (or you should add) a vercel.json file inside the Backend directory. This file tells Vercel how to build and route the backend. For example, a minimal *vercel.json* for an Express server could be:

```
{
 "builds": [
  { "src": "index.js", "use": "@vercel/node" }
 ],
 "routes": [
  { "src": "/(.*)", "dest": "/" }
 ]
}
```

This configuration says: use the *@vercel/node* runtime to build the *index.js* entry, and route all requests to this serverless function. Ensure that the main file name (index.js in this example) matches your actual entry file (if your main file is app.js, put that in the src).

- Alternative (if no vercel.json): Name your main server file *index.js*. Vercel by default looks for index.js in the project root for Node projects. If found, it will automatically deploy it as a serverless function. If your file is named differently, either rename it or use vercel.json as above.
- Set the Build Command to npm install (since for a Node API, you basically just need to install dependencies; the *@vercel/node* builder will handle bundling). You likely don't need a separate build step unless you have a compilation (like TypeScript or Babel) – if so, ensure the build command does that.
- The Output directory is not applicable for serverless functions (Vercel will handle the output).
- Environment Variables: Add all the required backend environment variables in this project's settings (MONGO_URI, OPENAI_API_KEY, etc. – everything listed in Section 6.1 except PORT). You can do this before deploying (via the same interface, there's an Environment Variables section during project setup). Or add them immediately after deploying in the project's Settings > Environment Variables. This step is crucial – without these, the backend will fail to connect to database or other services.
- Deploy the Backend: Click "Deploy". Vercel will then install dependencies and package the Node.js function. If configured correctly, it will output a success. If it fails, check the logs:
    - Common issues: missing environment variables (leading to connection failure), or not finding *index.js* (meaning vercel.json config is likely needed or incorrect).
    - Ensure that the Express app is listening to the correct port. In Vercel's serverless environment, you do not manually specify a port; the runtime will handle it. Your Express app should use *process.env.PORT* if available. In our code, we had *PORT=8081* for dev, but on Vercel that env var might be overridden or not needed. Typically, you can modify the code to use *process.env.PORT || 8081*. Vercel will set a port when running the function. The dev PORT in .env is ignored on Vercel (since we don't manually listen on 8081 in a cloud function).
    - If needed, remove the explicit port or allow it to default. The dev.to guide we referenced shows using *process.env.PORT || 3000* in code which is a good practice for compatibility.
- Test the Backend Endpoint: Once deployed, you'll get a domain like *https://codesaurus-backend.vercel.app*. You can test the backend by hitting an endpoint. For example, try a GET request to the base URL or */api/your-endpoint* using a browser or curl. If the backend returns expected data or a welcome message, it's running. If you get a Vercel 404 or 500 error, check the routing:

        - A 404 might mean your vercel.json routes are not set correctly or the function didn't deploy. Verify that vercel.json is present in the deployed code (the build log should mention using @vercel/node).
        - A 500 could indicate an error in the code (check Vercel function logs under the Functions tab or Logs tab for your project).
    - Enable CORS (if not already): In production, your frontend will be on a different domain than the backend (e.g., codesaurus-frontend.vercel.app calling codesaurus-backend.vercel.app). This is cross-origin. Ensure the backend has CORS enabled to accept requests from the frontend domain. If you hadn't done so, you can quickly enable it by using the cors middleware in Express. For example:

      const cors = require('cors');

      app.use(cors({

      origin: '' // or specify the exact origin: 'https://your-frontend.vercel.app'*

      }));

      This should be done in code and redeployed. Without this, the browser will block the API calls in production (they'll appear as CORS errors in the browser console).

- Reconnect Frontend to Backend: Now that both are live, update the frontend configuration to point to the backend:
    - If you set REACT_APP_API_URL in the frontend Vercel settings, ensure its value is the full URL of the backend (e.g., *https://codesaurus-backend.vercel.app*). If you hadn't set it yet, do it now: go to the frontend project's settings on Vercel, add REACT_APP_API_URL with the backend URL, and redeploy the frontend. This will bake the URL into the production build.
    - If the frontend was using relative paths or proxy in dev, those won't work in production. So this step is important to tell the frontend where to send requests.
    - Redeploy the frontend (you can trigger redeploy by clicking "Deploy" or making a dummy commit to GitHub if it's connected).
- Final Verification: Visit your frontend's Vercel URL (e.g., your-frontend-project.vercel.app) in the browser. Now try using the app as an end-user:
    - The app UI should load.
    - When you perform actions (login, story progression), it should now be making requests to the backend Vercel URL. Check the browser's network requests – they should be going to codesaurus-backend.vercel.app (HTTPS). They should succeed (status 200). If you see CORS errors or failed requests, re-check that CORS is enabled on the backend and that the frontend is pointing to the correct URL.

- Ensure data is saving to the cloud DB (check MongoDB Atlas for new entries created via the production app).
- Check that images upload to S3 from the production app (if applicable). Because now the requests to AWS will come from Vercel's servers – the IAM user permissions should allow that, since AWS doesn't restrict by source IP by default when using access keys.
- Basically perform the same tests as in Section 7 but on the production URLs.

Both the frontend and backend are now deployed on Vercel. They will each have their own default subdomain. Users of your app will primarily interact with the frontend URL, which under the hood calls the backend URL for data.

Using Vercel Default Subdomains: We have used Vercel's provided subdomains (the *.vercel.app URLs). These are secure (HTTPS by default) and publicly accessible. You can share the frontend URL with testers or users to try out the application. The backend URL doesn't necessarily need to be directly accessed by users (it's consumed by the frontend), but it's a good idea to keep it secure or not widely publicized. If desired, you could add basic authentication or obscurity to the backend, but usually a well-configured CORS is sufficient to control access (so only your frontend uses it).

At this stage, your full stack application is live. In the next section, we will discuss troubleshooting common deployment issues and how to address them.

9. Troubleshooting

Despite careful setup, you may encounter issues either during deployment or while running the application. Below are common problems and their solutions, categorized by backend, frontend, and deployment issues.

9.1 Backend Issues

- Backend Not Starting Locally: If npm start (or node index.js) doesn't start the server, ensure that:
  - You ran npm install in the Backend folder and it completed without errors.
  - The Node version meets the requirement (Node 16+). You can check package.json for any engine requirements.
  - There are no syntax errors or missing module errors when running. If there are, install missing packages or fix code issues.
  - The .env file is present and parseable. A missing .env might cause the app to crash if it expects certain vars. Add fallback checks in code or provide all required env vars.
- Cannot Connect to MongoDB (locally or on Vercel): This often shows up as a timeout or authentication error in logs.
  - Double-check the MONGO_URI. If it contains special characters in the password, ensure they are URL-encoded.

    - Make sure your IP or Vercel's IPs are allowed by Atlas. For testing, set Atlas network access to 0.0.0.0/0 (all IPs) if you haven't.
    - Verify that the Atlas cluster is running (sometimes free clusters pause when not in use; access the Atlas UI to wake it).
- OpenAI API Errors: If the backend logs an error when calling OpenAI (or the feature doesn't work):
    - Check that OPENAI_API_KEY is correct and has not expired or been revoked.
    - Ensure you haven't exceeded OpenAI rate limits or quota. The error messages from OpenAI can indicate if that's the case.
    - The request might be malformed or missing parameters. Use console logs or OpenAI's dashboard to debug request usage.
- AWS S3 Upload Errors: If images or files aren't uploading/retrieving:
    - Check the error message. If it's "Access Denied", the IAM user likely lacks proper S3 permissions. Adjust the IAM policy to allow the actions needed (PutObject, GetObject on the bucket).
    - If it's "No Such Bucket", verify S3_BUCKET_NAME exactly matches the created bucket name.
    - If using a region-specific endpoint, ensure AWS_REGION is correct.
    - Try a manual upload using AWS CLI with the same keys to see if that works, to isolate if it's code or credentials.
- CORS errors on Backend (during development): If the frontend (localhost:3000) cannot fetch from backend (localhost:8081) due to CORS:
    - Implement CORS middleware in backend for dev: e.g. app.use(require('cors')()); to allow all origins (or at least localhost:3000).
    - Alternatively, ensure the React dev proxy is configured. In frontend/package.json, a proxy setting to http://localhost:8081 can automatically handle CORS during dev by proxying requests.

9.2 Frontend Issues

- Frontend Failed to Compile: On running npm start (development) if the app doesn't compile:
    - Look at the error output in the terminal. Common issues include syntax errors in JSX/JS, or a missing environment variable (some React apps expect certain env vars to build).
    - Install any missing dependencies. If error says module not found, try npm install <module_name>.
    - If using a newer JSX transform, ensure your Node and npm versions are compatible with the React scripts version.
- Blank Page or Crash on Frontend: If the dev server opens but you see a blank page or error in console:
    - Open the browser's developer console (F12) and check for runtime errors (red errors). Trace those to components or missing config.

    - A common mistake is forgetting to configure something like API URL, causing fetch calls to fail and maybe not handled. Ensure any required configuration for the front is done.
    - Another cause can be incorrect routes. Check if the React app is trying to fetch an asset or route that doesn't exist, leading to an unhandled exception.
- Frontend Cannot Reach Backend in Production: After deploying, if the app loads but no data comes (and you see network errors in console):
    - Likely the API calls are failing. Check if they are going to the correct URL. If you see calls to localhost or to a wrong domain, the frontend isn't configured with the production API URL. Set the correct REACT_APP_API_URL and redeploy.
    - If the URL is correct but you get a CORS error (visible in browser console as blocked by CORS policy), the backend isn't allowing the frontend origin. Implement CORS on backend to allow the frontend's domain.
    - If you get 5xx errors from the API calls, check the backend logs on Vercel. It could be an unhandled exception on the server side.
- Styling or Asset Issues: If some CSS or images aren't showing:
    - Make sure static assets were built and deployed. Check the Vercel build output for any warnings about assets.
    - Verify the links in the HTML are correct (the React app should handle this, but if you see 404s for CSS/JS, might be a misconfiguration in routing).
    - Possibly, if using a base path, ensure the app knows if it's not at root. Usually not an issue with direct domain.

9.3 Deployment Issues

- Vercel Deployment Failing: If either project fails to deploy (you see a red cloud or error message in Vercel):
    - Check the build logs on Vercel for clues. For the frontend, any npm/yarn errors? For example, maybe a package locked to an old Node version not supported on Vercel's default Node (which is usually Node 18). You might specify a Node version in package.json "engines" if needed, or adjust the code.
    - For backend, if the log says it cannot find a module or file, ensure that all files are pushed to repo and that vercel.json is included if used. Also check case sensitivity (on Vercel's Linux environment, file names are case-sensitive).
    - If environment variables are missing, the backend might crash at runtime – Vercel might still show a "successful" deployment but the function returns 500. To debug, use Vercel's Functions logs: go to your backend project, "Functions" tab, and see logs for recent invocations.
- Environment Variables Not Taking Effect on Vercel: If it seems your app isn't reading the env vars (e.g., still connecting to wrong DB or failing at connecting):

    - Ensure you added the vars under the correct environment (Production) and then triggered a redeploy after adding. Adding a variable alone doesn't rebuild the app; redeploy to apply them.
    - Check that the variable names match exactly (typos or wrong case will mean process.env.X is undefined).
    - In the backend, print process.env keys (temporarily for debugging) to confirm presence (but do not print sensitive values).
    - On the frontend, if using a public var, check the built code or network calls to see if the value is reflected.
- Incorrect Domain or URL Configurations:
    - If you accidentally set the wrong URL for API and deployed, you might get failures. To fix, correct the environment variable and redeploy the frontend.
    - If you named the projects differently than expected, update any references. For instance, if the backend's actual domain is different, update the frontend config accordingly.
- Application Errors in Production Only: Sometimes an app works locally but not in production due to differences:
    - One common difference is case sensitivity in file paths (Windows/macOS file system is case-insensitive, but Linux (Vercel) is not). If your code imports ./Models/User.js but the file is models/User.js, it might work locally and fail on Vercel. Check your import paths and file name casing.
    - Another difference: environment. Vercel is Node 18 by default (at time of writing). If your code uses an older Node API or some experimental feature not available, adjust accordingly.
    - Timing or performance: serverless functions have a limited execution time. If an OpenAI call or some processing takes too long, the function might time out. Consider optimizing calls or using streaming if needed. Also, Vercel free tier has memory limits; very memory-heavy operations could potentially cause issues (not likely in this app unless large images in memory).
- Logs and Monitoring: Use Vercel's logging to monitor runtime errors. For the backend, every request invocation can be inspected. For the frontend, any console logs won't appear in Vercel (since it's static), but you can use browser dev tools to debug.

By systematically addressing issues as above, you should be able to overcome most common problems. Remember that deployment issues often boil down to configuration mismatches between dev and prod or missing dependencies. Double-check steps if something is not working. Finally, if you are stuck on an issue that's not producing helpful error messages, try to reproduce it locally if possible (for example, simulate production by setting NODE_ENV=production and running a build, or test with a local MongoDB if Atlas seems to be an issue, etc.). And consult community forums or the Vercel documentation for specific errors – often someone has encountered a similar deployment issue.

With troubleshooting covered, you have a comprehensive understanding of deploying 2025S-Codesaurus. By following this manual, you can confidently set up the system requirements, configure all necessary services, run the application locally, and deploy both the frontend and backend to Vercel's cloud platform using default subdomains. Enjoy your interactive storytelling platform.

10. Contact Information

For further assistance, please reach out:

Support Email: codesaurus25@gmail.com

GitHub Repository: https://github.com/htmw/2025S-Codesaurus.git

We're here to help! For immediate support, raise an issue on our GitHub page or contact us via email.