# AIrbnb: Exploring AI-Generated Property Listings

Niklas Berby
*Høgskolen i Østfold*
niklab@hiof.no

***Abstract*—This paper presents AIrbnb, a proof-of-concept platform that leverages large language models to generate immersive virtual property listings. We explore the intersection of artificial intelligence and digital marketplaces, demonstrating how AI can create engaging, fictional yet plausible travel destinations. Our implementation combines Preact for the frontend, Bun for the backend, and the OpenAI API for content generation, showcasing the potential of AI in creative content generation for e-commerce platforms.**

## I. Introduction

The rise of AI has created new possibilities in how we create and interact with web content. This project combines several modern technologies to create a unique property marketplace where all listings are generated by AI. The goal was to explore how AI could be used to create engaging content while learning about full-stack web development.

### A. Motivation

- Growing interest in full-stack web development
- Newfound interest in using Bun as a Typescript runtime
- Educational value in exploring AI capabilities in creative content generation

## II. System Design

### A. Architecture

a) *Frontend architecture:* The frontend is built using Preact, with a focus on component-based design and state management. Preact was chosen for its lightweight nature and compatibility with React components. Tailwind CSS is used for styling, providing a responsive and visually appealing user interface. It offers ease of use and flexibility in styling components.

While generation of new listings is handled by the backend API, the frontend will dynamically update to reflect the generated results.

b) *Backend architecture:* The backend is implemented using Bun, a lightweight Typescript runtime. It serves as the API layer for the frontend, handling requests for listing data and content generation. The backend also manages static assets and error handling. Bun was chosen primarily for its ease of use as well as my interest in exploring its capabilities.

The backend supports static serving of images generated by the AI model.

### B. Listing Generation

The project takes use of two different AI models—one for text generation and one for image generation. The text generation model is used to create engaging and immersive descriptions of the properties, while the image generation model is used to create visual representations of the generated listings.

The models used for content generation reside within the OpenAI API, specifically the GPT-4o-mini model for text generation and DALL·E 2 for image generation. These models work together to create listings that are both fictional and plausible.

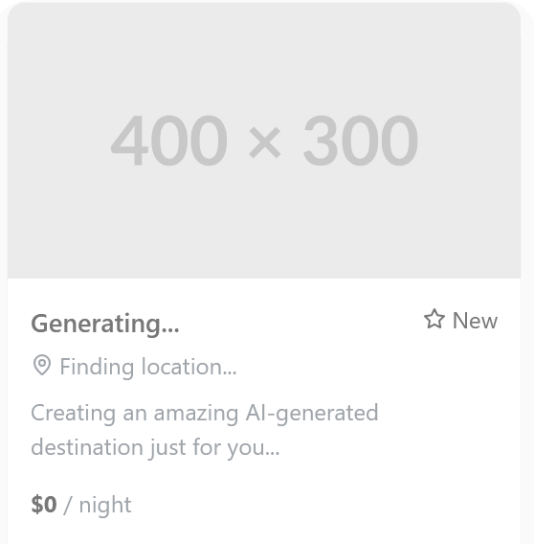


Fig. 1: Placeholder card while generating the listing.

### C. User Interface

The user interface design prioritizes immediate feedback and progressive enhancement while maintaining visual consistency with established marketplace patterns. Preact is used to create a responsive and interactive user interface that allows users to browse and interact with the generated listings.

a) *Layout Architecture:*

The interface employs a responsive grid system with three distinct breakpoints:

```
Mobile  (<768px):  Single column
Tablet  (≥768px):  Two columns
Desktop (≥1024px): Three columns
```

This progressive enhancement approach ensures content accessibility across devices while optimizing for each viewport's characteristics. The layout hierarchy follows:

1) Navigation and search (fixed position)
2) Main content grid (flexible)
3) Generation controls (floating)

Tailwind CSS utility classes are used to style components, providing a consistent visual language and responsive design.

## III. Implementation

### A. Frontend Development

To kickstart the project, I asked Claude 3.5 Sonnet to create a project scaffolding with placeholder images. This allowed me to focus on the backend and AI integration while having a visual representation of the final product. My initial prompt was:

> I want to create an Airbnb-like website as a personal project to showcase AI capabilities. Create a Preact project scaffolding for a website that lists various AI generated locations, but use image placeholders for now.

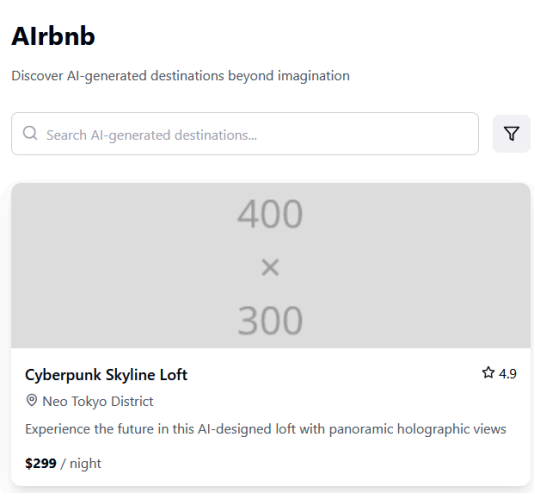


Fig. 2: Initial project scaffolding with placeholder images.

The model output a `.tsx` file with the structure described in Section II.C. Three cards were generated, each representing a fictional property listing. The cards included various fields such as title, description, and image placeholders. This output came fully equipped with Tailwind CSS classes for styling, showcasing the flexible grid system and responsive design.

Components were made with state management in mind, allowing for dynamic updates when new listings are generated. The search bar and filter options were not implemented, but they provided a clear path for future development.

### B. Backend Services

The backend services comprise three main files, each serving a specific purpose:

```
index.ts (server entry point)
generateListing.ts (content generation)
generateImage.ts (image generation)
```

While very feasible, the backend server was not made leveraging AI. I took this as a learning opportunity to explore Bun, a lightweight Typescript runtime. Bun offers a simple HTTP server, as well as an SQLite implementation, forgoing the need for additional dependencies. Documentation for the Bun HTTP functionality can be found at https://bun.sh/docs/api/http.

The server was set up to handle requests for listing data and content generation. It also served static assets and handled error responses. The server was designed to be lightweight and efficient, focusing on the core functionality of the project. Refer to `project/backend/src/index.ts` for the server implementation.

a) *Static asset server:* The server uses Bun's built-in static file serving to serve AI-generated images from the `project/backend/img` directory through the `/img/` route. Caching is enabled to reduce server load.

b) *API Endpoints:*

```
/api/
├── listings          [POST]
├── listings          [GET]
```

Details of the API endpoints can be found under Section VII.

### C. AI Integration

Discussion of the OpenAI API integration:

- Model selection and configuration
- Response processing
- Error handling
- Cost optimization

The prompts used for content generation can be found under Section VIII. The AI models used for text and image generation were selected based on their capabilities and compatibility with the project requirements. Both models were chosen with their low cost in mind, as the project is fairly small in scope, and my personal budget is limited.

The prompts themselves were generated by Claude 3.5 Sonnet, providing a clear structure for the content generation process.

a) *Data Flow:*

1) User requests new listing generation
2) Frontend displays optimistic UI update[1]

3) Backend initiates OpenAI API call
4) Response is parsed and validated
5) New listing is stored and returned
6) Frontend updates with actual data

## IV. Evaluation

### A. Performance Metrics

a) *Latency:* The models exhibit notable performance disparities between text and generation tasks. While text processing maintains acceptable latency, generation tasks demonstrate significantly longer response times. Perhaps the frontend could update the listing with partial information while waiting for the image generation to complete.

b) *Generation Quality:*

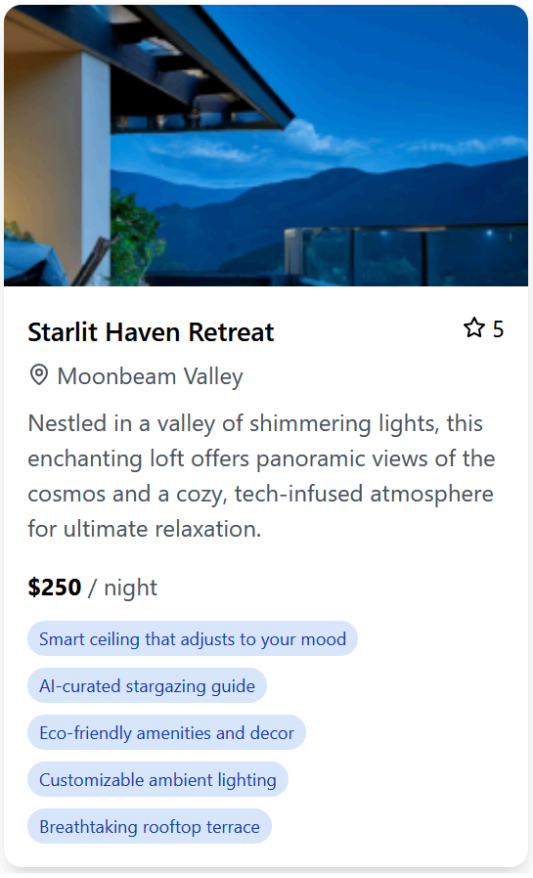


Fig. 3: Generation example.

The current implementation's generation quality shows limitations that can be traced to the use of cost-optimized models. While this approach achieves budget efficiency, it results in several quality compromises:

- Reduced output fidelity compared to premium models
- Less consistent generation results
- Limited handling of complex or nuanced requirements

These limitations are deemed acceptable for the project's scope.

### B. User Experience

While the interface provides a seamless user experience with responsive design and immediate feedback on listing generation, it is fairly limited. A page with more detailed information about each listing—including a larger image and additional details—would enhance the user experience.

### C. Limitations

This proof-of-concept project merely demonstrates the potential of AI-generated content. More in-depth iterations could explore advanced AI models, additional features, and user interactions to create a more engaging platform. I have purposefully kept the project scope small to focus on the core functionality.

## V. Conclusion

This project has been a valuable learning experience in both full-stack web development and AI integration. I have successfully implemented a proof-of-concept platform that generates fictional property listings, demonstrating the potential of AI in creative content generation for e-commerce platforms.

## VI. Appendix A: Setup Instructions

### A. Prerequisites

List of required software and dependencies, including:

- Bun
- SQLite
- OpenAI API key (for generation only)

By default, the OpenAI library looks for an environment variable named `OPENAI_API_KEY` to authenticate requests. This key is not provided in the repository, so you will need to set it up manually. The project has not been tested on an expired or invalid API key.

### B. Installation

```
# Install backend packages
cd backend/
bun install
# Run the backend server
bun src/index.ts
```

The server will be available at `http://localhost:3000`.

```
# Install frontend packages
cd frontend/
bun install
# Run the frontend server (development mode)
bun dev
```

[1] An optimistic UI update is a UI change that occurs immediately, before the backend has confirmed the action. This provides a more responsive user experience.

The frontend will be available at `http://localhost:5173` (default Vite port).

## VII. Appendix B: API Documentation

```
/*
Queries the database for all listings and returns them as JSON.
Should return an array of Listing objects, each containing property details.
Does not do any external API calls.
*/
GET /api/listings

/*
Generates a new property listing using the OpenAI API and adds it to the database.
Both the text and image data are generated by the AI model.
Returns the generated listing as a JSON object.
*/
POST /api/listings
```

## VIII. Appendix C: Prompts

### A. *Listing Generation Prompt*

"You are a creative travel destination generator specializing in imaginative, AI-generated locations. Generate a rental listing with the following characteristics:

Each listing should feel both fantastical yet somehow plausible, include subtle references to AI and technology without making it overwhelming, and balance wonder with practicality.

Please generate a listing using the following JSON format:

```
{
  "title": "A captivating name for the property (max 40 characters)",
  "location": "An evocative location name (max 30 characters)",
   "description": "A compelling 2-3 sentence description (max 200 characters)",
   "propertyType": "The type of accommodation (e.g., loft, villa, treehouse)",
    "keyFeatures": ["Array of 3-5 standout features"],
  "pricePerNight": "A number between 100-1000",
   "idealFor": "1-2 target guest types (e.g., 'Digital nomads', 'Tech enthusiasts')"
}
```

Ensure the response is valid JSON and includes all fields. Balance innovation with homey comfort and avoid dystopian elements or unrealistic features."

### B. *Image Generation Prompt*

"Generate a realistic image of a housing listing on Airbnb. The image must not be abstract - it is housing for real people. Description: {*listing.description*}[2]"

[2]The generated listing description will be provided by the text generation model.