

Customify: An AI-Powered Interior Design Platform for Photo-Based Room Redesign, Itemised Cost Estimation, and 2D Blueprint Generation

Ch Koushik^{1,*}, K Surya Pavan², N Sai Ganesh³, and B Yuvaraj⁴

^{1,2,3,4}Department of Computer Science & Engineering (AIML), Parul University, Vadodara, Gujarat, India

¹2303031460028@paruluniversity.ac.in, ²2303031460082@paruluniversity.ac.in,

³2303031460250@paruluniversity.ac.in, ⁴2303031460221@paruluniversity.ac.in

Under the Guidance of: **Prof. Ritu Agarwal**, Department of Computer Science & Engineering (AIML), Parul University

*Corresponding Author: Ch Koushik

Abstract—Redecorating a home is costly, time-consuming and hard to follow for the average family: to preview a new look, the owner must engage a designer, and then separately price the furnishings and hunt for suitable products. We introduce **Customify**, a full-stack web application that takes one photograph of a room and produces (i) a redesign preview conditioned on a chosen style, (ii) an item-by-item Indian Rupee cost estimate over three budget bands, (iii) hand-picked buying links on Indian online marketplaces, and (iv) a floor-plan blueprint that can be downloaded as a PDF. The system is split into a React.js 18 single-page front end, a FastAPI service layer protected by bcrypt-hashed passwords and JSON Web Tokens, a SQLAlchemy data layer (SQLite for development, PostgreSQL for production), and four domain modules covering image generation, costing, product recommendation and blueprint drawing. Images are produced with Stable Diffusion img2img via the Replicate API whenever a token is available; without one, a deterministic colour-grading routine takes over, so the application keeps working offline. Testing the implementation, we measured a mean end-to-end response time of 155 ms for 1080p uploads and 443 ms for 4K uploads, a mean blueprint rendering time of 36 ms for PDFs of roughly 3.6 KB, and correct behaviour in every authentication and validation check. The price engine holds exact entries for 35 of 35 furniture classes under seven style multipliers. Two shortcomings surfaced during testing; chiefly, the heuristic room-type detector returns “living room” in every case unless the user provides a hint.

Index Terms—Interior Design, Generative AI, Stable Diffusion, Image-to-Image Translation, Cost Estimation, Floor-Plan Generation, FastAPI, React.js, JSON Web Tokens, Recommendation Systems.

I. INTRODUCTION

Furnishing and decorating a home ranks among the biggest optional expenditures a family makes, and picturing the finished room is the first and most difficult stage. The usual route involves meeting a designer, going through mood boards, waiting for 2D or 3D renders, and only then being handed a price list. This takes anywhere from days to weeks and is beyond the reach of many middle-income households, especially in tier-2 and tier-3 towns where few professional designers work. At the same time, diffusion-based image synthesis [1, 2] has reached the point where a photograph can be reimagined in a matter of seconds, which hints that the visual step can be automated and combined with the cost and shopping details a buyer actually needs.

A. Structural Shortcomings in Existing Tools

Three weaknesses recur across tools aimed at consumers:

1. **Pictures without prices.** General text-to-image generators yield appealing images but say nothing about the cost of the furniture shown or where it can be purchased, so inspiration never turns into action.
2. **Catalogues that ignore the user’s room.** Retailer catalogues and room planners let people browse products, yet they do not begin with the customer’s *own* photo; the outcome therefore ignores the real room and its style.
3. **No plan-level output.** Neither category yields a scaled floor plan that a carpenter, contractor or designer can take as a basis for measuring and discussing the layout.

B. Motivation

A practical redesign assistant has to join four artefacts that normally come from four different professionals: a visual concept, a bill of quantities, a shopping list and a plan. Customify generates all four from a single upload and keeps them in each user’s personal history, with the aim of shrinking the decision process from days to seconds and opening early-stage interior planning to people without expertise.

C. Problem Statement

The goal is to build, deploy and empirically test a web platform that receives a room photo and a style choice and returns a redesign preview, a multi-tier itemised cost estimate in INR, product purchase links and a 2D blueprint, while offering secure multi-user accounts and a lasting design history.

D. Core Contributions

- **Complete redesign pipeline.** One authenticated call handles upload checking, room analysis, image synthesis, costing and storage, and answers with a structured JSON payload (Section IV).
- **Fallback-capable generator.** A two-mode image generator relies on Stable Diffusion img2img when it can and on a deterministic colour-grade and vignette routine when it cannot, so demonstrations and development never hinge on outside services.
- **Three-band INR cost engine.** A price table of 36 entries combined with style multipliers ($0.85 \times - 2.8 \times$) yields per-item and total minimum/maximum figures for budget, mid-range and premium bands.

- **Code-drawn 2D blueprint.** A vector PDF writer lays out walls, a door, a window, to-scale furniture footprints, dimension arrows and a legend for six room types.
- **Transparent empirical review.** We report measured latency, output fidelity and cost coverage, along with a frank list of faults uncovered while testing.

II. BACKGROUND STUDY & LITERATURE SURVEY

A. Diffusion Models and Image Editing

Denoising diffusion probabilistic models are trained to undo a step-by-step noising process and deliver high-quality samples [1]; later work showed them outperforming GAN generators in image quality [7]. Rombach et al. shifted the diffusion process into the latent space of a pretrained autoencoder, which made high-resolution synthesis feasible on ordinary hardware and led to Stable Diffusion [2]. For modifying an existing picture, SDEdit adds noise to the input up to a middle timestep and then denoises it under a new condition; the coarse arrangement survives while the appearance changes [3], and this is the idea behind img2img. ControlNet supplies spatial conditions such as edges, depth or segmentation maps to hold structure more firmly [4], while InstructPix2Pix carries out edits described in plain language [5]. CLIP learns a shared image–text embedding space on which text conditioning rests [6].

B. Indoor Scene Understanding

Datasets such as Places [8] and ADE20K [9] make it possible to train classifiers that name a photo's room category (bedroom, kitchen, living room) and segment furniture. RGB-D collections, including SUN RGB-D [10] and Stanford 2D-3D-S [11], enable work on room layout and object-level analysis. Furniture detection is usually built from real-time detectors like YOLO [12] and deep residual backbones [13].

C. Recommendation and Cost Estimation

Recommending consumer products is a well-developed area [15, 16]; content-based and knowledge-based methods connect explicit user requirements (here, style and budget) to catalogue entries. In interior fit-out, costs are normally worked out by quantity surveyors from rate tables, an approach that closely resembles a lookup-then-multiply model whose output grows with finish quality.

D. Web Platform Security and Architecture

Single-page applications commonly use stateless sessions based on JSON Web Tokens [17] together with adaptive password hashing through bcrypt [18]. The REST architectural constraints [19] shaped our API, and SQLAlchemy [20] gives a database-independent persistence layer.

III. RESEARCH GAP & NOVEL CONTRIBUTIONS

A. Gap Analysis

Putting the surveyed work together reveals four gaps that matter for practical interior-design support.

- **Image generation is separated from buying.** Studies of diffusion-based editing [3, 4, 5] concentrate on picture quality and do not convert the result into a priced, purchasable list of furniture.
- **No plan-level deliverable.** Scene-understanding research produces labels or layouts [10, 11], not a scaled plan the user can download.
- **Reliance on heavy infrastructure.** Most prototypes need GPUs or paid APIs, which rules out classroom demos, offline development and low-resource settings.
- **Localisation.** Cost models and marketplaces seldom reflect Indian price levels (INR) and Indian retailers.

B. Customify's Approach

Customify responds to these gaps with (1) one request that branches into generation, costing, recommendation and plan drawing; (2) a two-mode generator with a deterministic backup; (3) a tiered price model in INR with style multipliers; and (4) a vector blueprint produced in code, all exposed through a protected and documented REST API.

IV. PROPOSED WORK & SYSTEM ARCHITECTURE

A. Four-Tier Architecture

Customify uses a decoupled client–server layout (Fig. 1):

1. **Presentation tier.** React.js 18 with React Router, Axios, React Dropzone and React Hot Toast. The screens are Landing, Login, Register, Dashboard, Design Studio and History. An `AuthContext` holds the JWT and attaches it to every API call. The client accepts only JPEG, PNG or WebP files of at most 10 MB.
2. **API tier.** A FastAPI application with ten REST endpoints under `/api/auth`, `/api/design`, `/api/blueprint` and `/api/products`; CORS is limited to the known front-end origins, and OpenAPI documentation is generated automatically at `/docs`.
3. **Service tier.** Four self-contained Python modules: `ai_service` (room analysis and generation), `cost_service`, `product_service` and `blueprint_service`.
4. **Data tier.** A SQLAlchemy ORM on SQLite (development) or PostgreSQL (production) with three entities: `User`, `Design` and `Product`. Uploaded and generated images live on disk and are served as static files.

The entire system is packaged with Docker Compose (the backend has a health check at `/health`; nginx serves the front end), so a reviewer can launch it with a single command.

B. Operational Workflow

A design request (Fig. 2) runs through these steps. (1) The user logs in and receives a JWT. (2) In the Design Studio the user drops in a photo, chooses one of seven styles and, if desired, a room type. (3) `POST /api/design/generate` checks the style and MIME type (using the file extension when the header cannot be trusted), stores the file under a UUID name and inserts a `Design` row with status *processing*. (4) `analyze_and_design` detects the room type, estimates the dimensions, picks a furniture list and produces the re-design. (5) The row is set to *completed* (or to *failed* together with the error text), and the cost engine returns the tiered estimates. (6) The client shows three tabs—Design, Cost

and Products—and the user can download the 2D blueprint, which is drawn on demand from the saved room type, dimensions and style.

C. Data Model

Fig. 3 depicts the three entities. A User owns many Design rows (with cascading delete), and a Design owns many Product rows. Design.detected.items keeps the furniture list as a JSON string, and status moves through the lifecycle *pending*→*processing*→*completed/failed*. Each read, delete and blueprint request is filtered by user_id, so people can reach only their own designs.

D. REST API Surface

Table I enumerates the endpoints. Every design, blueprint and history route needs a valid bearer token; a missing or invalid token produces HTTP 401.

V. METHODOLOGY & MATHEMATICAL FORMULATIONS

Customify was built step by step, one service at a time. Each service is a plain Python module that can be tested on its own and is then attached to the application through a router.

A. Room Analysis

For an image of $w \times h$ pixels, the platform guesses the room footprint in metres with a linear rule clipped to believable household dimensions:

$$W = \min\left(7.0, \max\left(3.0, 3.5 + 2.5 \frac{w}{1920}\right)\right), \quad (1)$$

$$H = \min\left(5.0, \max\left(2.5, 3.0 + 1.5 \frac{h}{1080}\right)\right), \quad A = W \cdot H. \quad (2)$$

The room type $r \in \mathcal{R}$, $|\mathcal{R}| = 6$ (living room, bedroom, kitchen, dining room, bathroom, home office), is copied from the user's hint when one exists; otherwise a brightness and aspect-ratio rule decides. The furniture list is a uniformly random subset of size $n \sim \mathcal{U}\{4, \dots, |P_r|\}$ taken from the pool P_r belonging to that room type.

B. Generative Redesign

Diffusion mode. When a Replicate token is present, the uploaded picture x_0 is sent as a base64 data URI along with a style prompt c_s (Section B), a negative prompt, guidance scale $g = 7.5$, 30 inference steps and prompt strength $\sigma = 0.65$. In img2img the forward noising step is

$$x_{t_0} = \sqrt{\bar{\alpha}_{t_0}} x_0 + \sqrt{1 - \bar{\alpha}_{t_0}} \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I), \quad (3)$$

where $t_0 \approx \sigma T$; the model then denoises with classifier-free guidance [3, 2]. A larger σ lets the result stray further from the source. The service checks the prediction every 2 s for as long as 120 s and then downloads the first output.

Mock mode. If no token is set, or the API call fails, a deterministic grade is applied instead. Let G be a Gaussian blur of radius 0.8, $h_s \in [0, 255]^3$ the style tint, $a_s = \alpha_s/255$ its opacity, and S_κ, B_β the saturation and brightness enhancers. The result is

$$I_{\text{out}} = V\left(\text{Sh}_{1.4}\left(B_{\beta_s}\left(S_{\kappa_s}\left((1 - a_s)G(I) + a_s h_s\right)\right)\right)\right), \quad (4)$$

in which V applies a 30-step vignette and a caption bar. The style-dependent parameters $(h_s, \alpha_s, \kappa_s, \beta_s)$ vary; *industrial*, for instance, uses $\alpha = 60$, $\kappa = 0.75$, $\beta = 0.95$, whereas *minimalist* uses $\alpha = 35$, $\kappa = 0.80$, $\beta = 1.12$.

C. Cost Estimation

Write $p_{i,\tau} = (\ell_{i,\tau}, u_{i,\tau})$ for the INR range stored for item i in tier $\tau \in \{\text{budget}, \text{mid}, \text{premium}\}$, and m_s for the style multiplier ($m_{\text{modern}} = 1.0$, $m_{\text{luxury}} = 2.8$, $m_{\text{bohemian}} = 0.85$, and so on). For a set D of detected items:

$$\text{Cost}_\tau(D, s) = \left[\sum_{i \in D} \lfloor m_s \ell_{i,\tau} \rfloor, \sum_{i \in D} \lfloor m_s u_{i,\tau} \rfloor \right]. \quad (5)$$

An item name is looked up by exact lowercase match first, then by substring, and finally falls back to a generic default range, so an unfamiliar item never raises an error. Purchase links come from a hand-built map, with an Amazon.in search URL created as the fallback.

D. Product Recommendation

Given a style s , room r and budget class b , the catalogue returns the items q with price range $[q_{\min}, q_{\max}]$ that pass

$$\text{keep}(q, b) = \begin{cases} q_{\max} \leq 18,000, & b = \text{budget} \\ q_{\min} \geq 20,000, & b = \text{premium} \\ \text{true}, & \text{otherwise,} \end{cases} \quad (6)$$

and, if nothing survives the filter, the first four entries for the style (or for *modern*) are returned instead.

E. Blueprint Rendering

The blueprint is drawn on an A4 landscape ReportLab canvas. If the usable area is $D_W \times D_H$ points and the room is $W \times H$ metres, the common scale is

$$S = 0.88 \cdot \min\left(\frac{D_W}{W+0.8}, \frac{D_H}{H+0.8}\right), \quad (7)$$

and a point (x, y) in metres maps to page coordinates $(M_L + 0.4 \text{ cm} + xS, M_B + 0.4 \text{ cm} + yS)$. Every room type has a fixed arrangement of coloured rectangles (x, y, w, h) in metres, drawn as rounded boxes over a 1 m grid. Walls take the style's accent colour, and the sheet also carries a door arc, a three-pane window, dimension arrows and a legend.

F. Authentication

Passwords are hashed with bcrypt [18]. Tokens are JWTs [17] signed with HMAC-SHA256, carrying `sub` = email and expiring after 7 days; the signing secret is read from the environment.

VI. EXPERIMENTAL RESULTS & DISCUSSION

A. Setup

The backend was run unmodified through FastAPI's in-process test client on a CPU-only cloud Linux container with Python 3.13, in the offline (mock) configuration and without a Replicate token. No labelled room photographs were on hand in the test environment, so performance runs used synthetic room-like images (flat-colour regions plus noise) saved as JPEG at four resolutions, with $N = 10$ requests per resolution cycling through the seven styles and

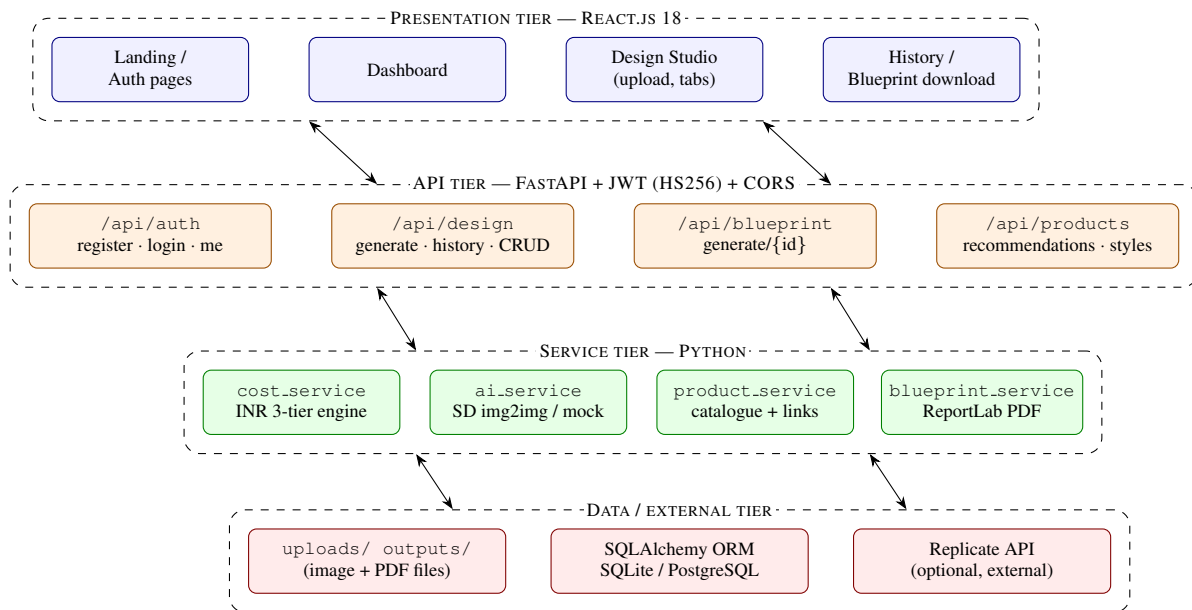


Figure 1: System architecture of Customify. A tier talks only to the tier next to it, and the Replicate service on the outside is optional.

Table I
REST API OF CUSTOMIFY

Method	Endpoint	Purpose
POST	/api/auth/register	Create account, return JWT
POST	/api/auth/login	Authenticate, return JWT
GET	/api/auth/me	Current user profile
POST	/api/design/generate	Upload + redesign + cost
GET	/api/design/history	List user's designs
GET/DELETE	/api/design/{id}	Fetch / delete a design
GET	/api/blueprint/generate/{id}	Download 2D blueprint PDF
GET	/api/products/recommendations	Style/room/budget products
GET	/api/products/styles	List the seven styles

Table II
DESIGN-GENERATION LATENCY (MOCK MODE, $N = 10$)

Input size	Mean (ms)	SD (ms)	p95 (ms)	Max (ms)
640×480	65.4	11.4	82.2	82.3
1280×720	109.6	11.6	128.2	137.5
1920×1080	155.1	27.2	204.1	208.9
3840×2160	443.4	65.0	536.5	585.2

six room types. The figures therefore describe system behaviour rather than perceptual quality, and network time is not counted.

B. End-to-End Latency

Table II and Fig. 4 give the time from upload to JSON reply (validation, disk write, database insert, analysis, generation, update and costing). Latency rises roughly in proportion to pixel count, since the generator shrinks the decoded image to at most 900×900 ; even a 4K image finishes in under 0.6 s. Registration took 329 ms and login 285 ms, mostly because of bcrypt's intentionally high cost factor. Blueprint generation averaged 36 ms ($\sigma = 52$ ms) with a mean file size of 3.6 KB.

C. Functional and Security Checks

We ran five behavioural checks. A wrong password gave HTTP 400; a call with no token gave 401; uploading a text file gave 400; an unrecognised style ("gothic") gave 400; and well-formed requests gave 200 with a full payload. All five outcomes agreed with the specification. Filtering by `user_id` on every query stops one account from reaching another's designs or blueprints.

D. Output Fidelity of the Offline Generator

Fig. 5 displays the seven style grades applied to a single synthetic room, and Table III measures how far each departs from the source: PSNR lies between 17.4 and 23.1 dB and the mean absolute error between 12.9 and 28.7 grey levels (0–255 scale). Warm styles (bohemian, rustic, luxury) slightly darken the image, while cool or bright styles (modern, minimalist, scandinavian) lighten it; industrial changes the image most. The numbers show that the offline mode yields clearly different, style-consistent grades, but we emphasise that it only simulates colour grading and *cannot* add or swap furniture—that ability belongs to the diffusion mode alone.

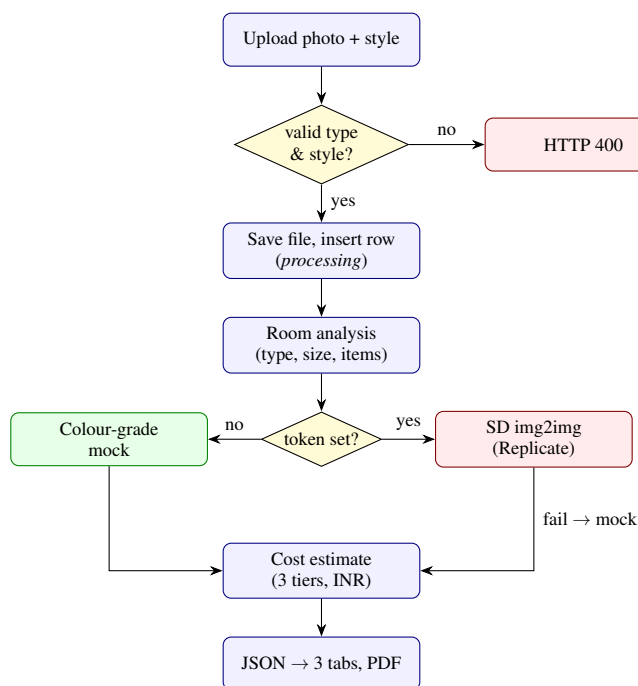


Figure 2: Request flow of the design-generation endpoint. If Replicate fails, the request quietly drops back to the mock generator.

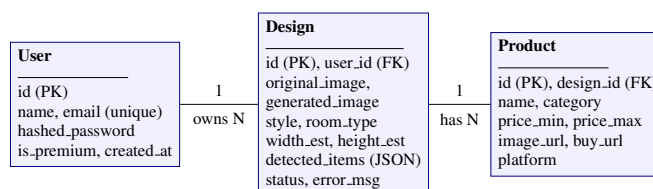


Figure 3: Entity–relationship diagram of the Customify database schema.

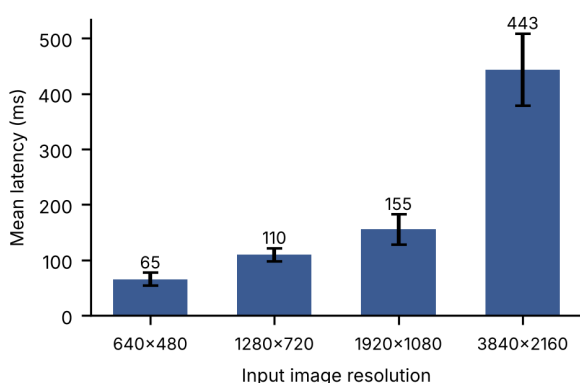


Figure 4: Mean generation latency against input resolution (error bars show one SD).

E. Cost-Engine Behaviour

For the eight-item living-room set, Table IV lists estimated totals in lakh INR (1 lakh = 100,000). The luxury multiplier scales every tier by 2.8 compared with modern, and bohemian comes out 15% cheaper. The tiers are contiguous: the lower limit of mid-range equals the upper limit of budget. Under the modern style, mid-range totals are Rs. 0.75–2.32 lakh (living room, 8 items), Rs. 0.89–2.49 lakh (bedroom, 7), Rs. 0.83–2.44 lakh (kitchen, 6), Rs. 0.65–1.96 lakh (dining room, 6), Rs. 0.95–2.53 lakh (bathroom, 6) and Rs. 0.41–

Table III
OFFLINE GENERATOR: DEVIATION FROM SOURCE IMAGE

Style	PSNR (dB)	Mean shift	MAE
Modern	21.7	+15.3	20.3
Minimalist	19.1	+22.0	25.9
Bohemian	23.1	−5.6	13.3
Scandinavian	20.4	+17.5	22.3
Luxury	22.5	−11.3	13.0
Industrial	17.4	−27.8	28.7
Rustic	23.0	−7.9	12.9

1.21 lakh (home office, 6). All 35 distinct furniture names the analyser uses found an exact match among the 36 price entries, but only 18 of the 35 carry a hand-picked purchase link; the other 17 use generated Amazon.in search links.

F. Blueprint Output

Fig. 6 shows the blueprint for a 5.2 m × 4.0 m modern living room, with furniture footprints drawn to scale on the 1 m grid, plus a legend and dimension arrows.

G. Catalogue Coverage

The product catalogue contains 30 curated items: 22 for living rooms spanning all seven styles, 4 for bedrooms and 4 for home offices (modern style only). Any other combination falls back to the living-room or modern lists. Blueprint layouts are defined for all six room types.

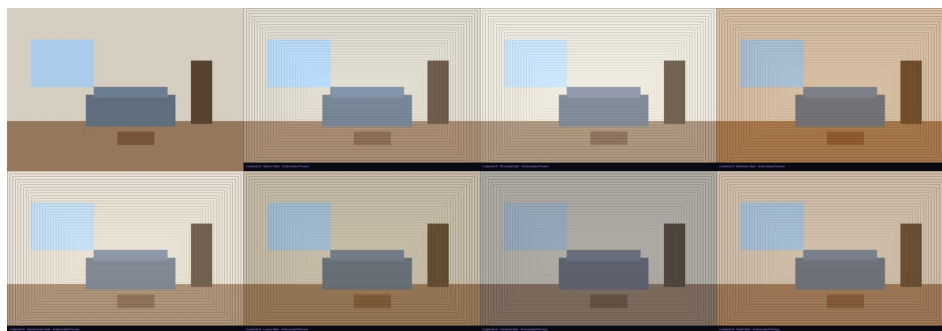


Figure 5: A synthetic room (top-left) together with the seven offline style grades the platform produces.

Table IV
LIVING-ROOM COST ESTIMATES BY STYLE (LAKH INR)

Style	Budget	Mid-range	Premium
Modern	0.24–0.75	0.75–2.32	2.32–8.45
Minimalist	0.22–0.68	0.68–2.09	2.09–7.61
Bohemian	0.20–0.64	0.64–1.97	1.97–7.18
Scandinavian	0.24–0.75	0.75–2.32	2.32–8.45
Luxury	0.67–2.10	2.10–6.50	6.50–23.66
Industrial	0.26–0.82	0.82–2.55	2.55–9.29
Rustic	0.23–0.71	0.71–2.20	2.20–8.03

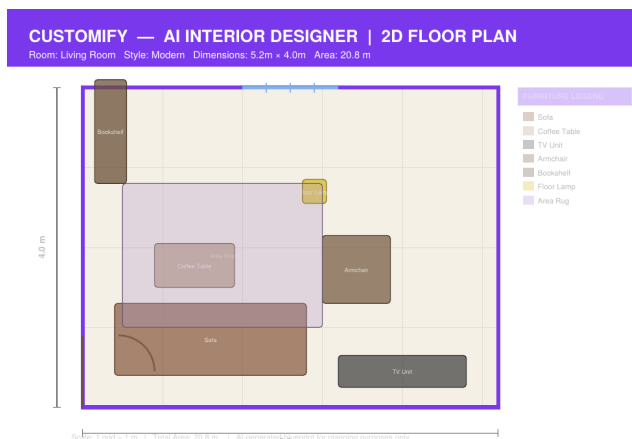


Figure 6: Automatically generated 2D blueprint (living room, 5.2 m × 4.0 m).

VII. COMPARATIVE ANALYSIS & ADVANTAGES

Table V sets Customify beside two classes of existing tools in qualitative terms. The comparison is based on features and kept generic; it is not a benchmark of specific products.

A. Core Advantages

- **One upload, four outputs:** preview, costs, links and plan.
- **Resilience:** generation never waits on an outside service; failures drop back to the offline mode.
- **Cheap to deploy:** a single Docker Compose command, SQLite by default, no GPU.
- **Localised:** INR prices and Indian marketplaces.
- **Secure by default:** hashed passwords, expiring tokens, per-user data separation and input checks.

VIII. LIMITATIONS

These limitations were identified through code review and testing.

1. **Room-type detection defect.** In `_detect_room_type`, a pixel from the lower crop is read at a y-coordinate beyond that crop, triggering an `IndexError`; the enclosing exception handler then answers “living_room”. Across 200 trials on one test image the answer never changed. The room type is right only when the user states it. A scene classifier trained on Places [8] would be a sound replacement.
2. **The offline mode is not generative.** It alters colour tone only and leaves the furniture untouched. Moreover, in both modes the furniture list (`_detect_furniture`) is a random draw from a fixed pool and not the output of object detection, so the cost estimate is indicative and not specific to the photograph.
3. **Dimension estimate.** Room size comes from the image’s pixel dimensions rather than from geometry; it is a stand-in for monocular layout estimation [11].
4. **Template-based blueprint.** Layouts are fixed for each room type and do not mirror the arrangement in the photo. Some detected items (for kitchens and bathrooms, say) have no match in the template: only 1 of 6 detected item names overlaps with the template for each of those rooms.
5. **Blueprint rendering artefacts.** Looking at the rendered PDF (Fig. 6) shows that the translucent fill of the area rug is never reset, so the legend, grid caption and footer appear faint, and the fixed door arc can run into furniture such as the sofa.
6. **Static prices.** The price table is hard-coded rather than live, the catalogue is small, and bedroom and office coverage exists only for the modern style.
7. **No perceptual evaluation.** We ran no user study and computed no learned perceptual metrics such as SSIM [14], FID or CLIP-score on real photographs; only synthetic inputs were used.
8. **Security hardening.** The default `SECRET_KEY` must be changed before deployment, the file-type check relies on headers and extensions rather than file signatures, and there is no rate limiting.

Table V
FEATURE COMPARISON BY TOOL CLASS

Capability	Generic text-to-image	Catalogue / planner apps	Customify
Starts from user photo	Optional	Rarely	Yes
Style selection	Via prompt	Via filters	7 presets
Itemised INR cost	No	Per product	3-tier, per item
Purchase links	No	Own store	Amazon / Flipkart
2D plan export	No	Varies	PDF
Accounts + history	Varies	Yes	Yes
Runs without GPU/API	No	N/A	Yes (mock)

IX. CONCLUSION & REALISTIC FUTURE WORK

We have described Customify, a full-stack platform that turns a room photograph and a style selection into a redesign preview, an itemised three-tier INR estimate, purchase links and a scaled 2D blueprint, all behind a protected and documented API. The implementation is light, containerised and robust: image generation switches to an offline mode on its own, and measured response time stays under 0.6 s even for 4K inputs. Testing also uncovered concrete faults, above all the room-type fallback bug and the randomly drawn furniture, and these set the near-term roadmap.

A. Realistic Future Directions

1. **Genuine perception.** Swap the heuristics for a scene classifier [8], a furniture detector [12, 9] and monocular layout estimation, so that room type, objects and dimensions are actually measured.
2. **Structure-preserving generation.** Use ControlNet-style conditioning [4] so the redesign keeps walls, windows and perspective, and align the generated furniture with the cost and blueprint outputs.
3. **Live commerce data.** Connect marketplace affiliate APIs for current prices and stock.
4. **Evaluation.** Carry out a user study and report SSIM [14], CLIP-based alignment [6] and FID on real room photographs.
5. **Production hardening.** Repair the blueprint alpha-state leak, default to PostgreSQL, validate uploads by file signature, add rate limiting and rotate refresh tokens.

ACKNOWLEDGEMENT

The authors are grateful to Prof. Ritu Agarwal and the Department of Computer Science & Engineering (AIML), Parul University, for their guidance during the design, implementation and evaluation of this work.

REFERENCES

REFERENCES

- [1] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 6840–6851.
- [2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proc. IEEE/CVF CVPR*, New Orleans, LA, USA, June 2022, pp. 10684–10695.
- [3] C. Meng et al., "SDEdit: Guided image synthesis and editing with stochastic differential equations," in *Proc. ICLR*, 2022.
- [4] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control

- to text-to-image diffusion models," in *Proc. IEEE/CVF ICCV*, Paris, France, Oct. 2023, pp. 3836–3847.
- [5] T. Brooks, A. Holynski, and A. A. Efros, "InstructPix2Pix: Learning to follow image editing instructions," in *Proc. IEEE/CVF CVPR*, Vancouver, Canada, June 2023, pp. 18392–18402.
- [6] A. Radford et al., "Learning transferable visual models from natural language supervision," in *Proc. ICML*, PMLR vol. 139, July 2021, pp. 8748–8763.
- [7] P. Dhariwal and A. Nichol, "Diffusion models beat GANs on image synthesis," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 8780–8794.
- [8] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 6, pp. 1452–1464, June 2018.
- [9] B. Zhou et al., "Semantic understanding of scenes through the ADE20K dataset," *Int. J. Comput. Vis.*, vol. 127, no. 3, pp. 302–321, Mar. 2019.
- [10] S. Song, S. P. Lichtenberg, and J. Xiao, "SUN RGB-D: A RGB-D scene understanding benchmark suite," in *Proc. IEEE CVPR*, Boston, MA, USA, June 2015, pp. 567–576.
- [11] I. Armeni et al., "3D semantic parsing of large-scale indoor spaces," in *Proc. IEEE CVPR*, Las Vegas, NV, USA, June 2016, pp. 1534–1543.
- [12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE CVPR*, Las Vegas, NV, USA, June 2016, pp. 779–788.
- [13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE CVPR*, Las Vegas, NV, USA, June 2016, pp. 770–778.
- [14] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.
- [15] F. Ricci, L. Rokach, and B. Shapira, Eds., *Recommender Systems Handbook*. New York, NY, USA: Springer, 2011.
- [16] C. C. Aggarwal, *Recommender Systems: The Textbook*. Cham, Switzerland: Springer, 2016.
- [17] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, IETF, May 2015.
- [18] N. Provos and D. Mazières, "A future-adaptable password scheme," in *Proc. USENIX Annual Technical Conf. (FREENIX Track)*, Monterey, CA, USA, June 1999, pp. 81–91.
- [19] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, CA, USA, 2000.
- [20] M. Bayer, "SQLAlchemy," in *The Architecture of Open Source Applications, Vol. II*, A. Brown and G. Wilson, Eds. 2012.