



Rajarata University of Sri Lanka
Faculty of Applied Sciences - Department of Computing
Project– ICT 3411 & COM 3405

Application for EC05

Section 01

Team Name:	Apexa
Project title:	Automated House Planning and Visualization System using Machine Learning
Git Hub Link:	https://github.com/Apexa-hub/2ndImplementationApexa.git

Section 02

Table of Contents

Section 01	1
Section 02	2
Section 03	2
Section 04	38
Section 05	45
Section 06	53

Section 03

Complete Research Design

3.1: Overview of Research Design

- **Shape Detection Model:** The shape detection model initiates the design process by allowing the user to input a survey plan. Through an interactive selection, the model enables the user to choose a specific slot from the survey plan. It then analyzes the selected slot, detects its geometric shape, and searches for the most suitable parcel that aligns with the slot's shape. Once a compatible parcel is identified, the model generates an appropriate building footprint within it, concluding the shape detection phase. This footprint serves as the foundation for subsequent steps in the project.
- **2D Floor Plan Generation Model:** With the generated building footprint from the shape detection model as input, the 2D floor plan generation model performs the task of room segmentation. The model divides the footprint into different rooms according to spatial and architectural principles, effectively creating a structured floor plan. Each room type (such as bedrooms, living rooms, kitchens) is represented with specific colors or labels to guide the model. The output from this model is a complete 2D floor plan, ready for conversion into 3D form in the final phase.
- **3D Visualization Model:** In the final stage, the 3D visualization model converts 2D binary wall mask images into 3D wall models by identifying black pixels (walls) in the image, creating cuboid meshes for each pixel, and stacking them in 3D space based on their positions in the image. Using OpenCV, it processes the image to detect walls, while Open3D generates and combines cuboids with specified height and thickness into a unified 3D mesh. The resulting 3D model, representing walls in the layout, is saved as a `.ply` file for architectural visualization or further use.

3.2: Overall Data set Preparation

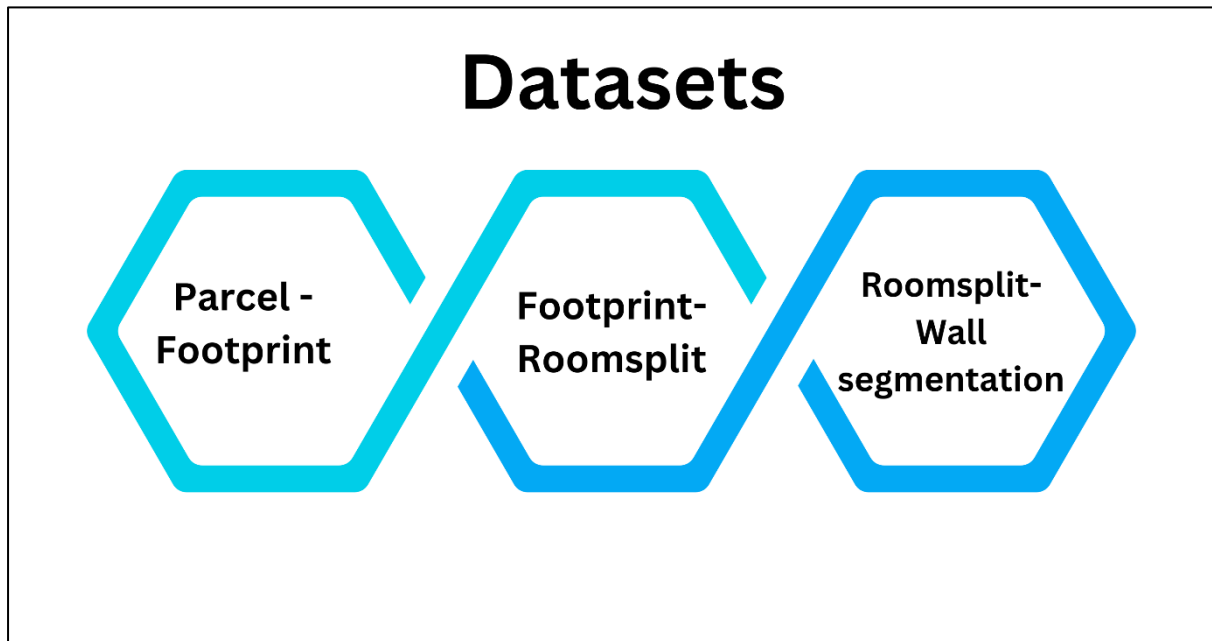


Figure 1: Prepared Datasets

- We have used three datasets for our model creation as mentioned above.
 - Parcels with annotated building footprints.
 - Building footprints with annotated room splits.
 - Room split with wall segmentation.

3.2.1. Parcels with annotated building footprints

- The dataset is created through the following process:
 - a) Obtaining Parcels from Survey Plans
 - Parcels are extracted from survey plans. These survey plans provide accurate geographical boundaries, which are crucial for the dataset.

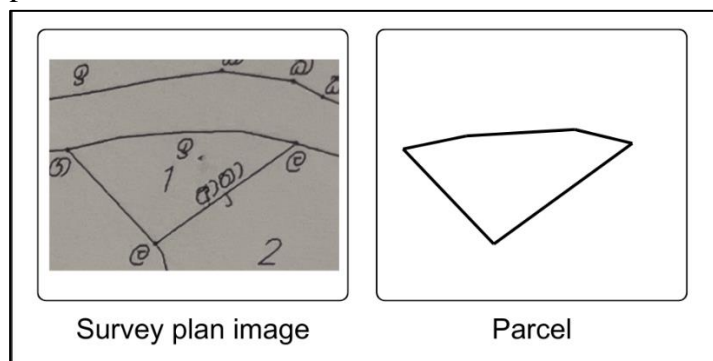


Figure 2: Capturing Slot from survey plan

- b) Annotating Building Footprints

- We utilize the Robin dataset, which contains floor plans of various buildings. The preparation involves:
- Resizing the Robin Dataset: To ensure consistency, all images from the **Robin dataset** are resized to a standard dimension (512*512px).
- Annotating Floor Plans: Each floor plan is manually annotated to extract building footprints. These annotations capture the structural layout of the buildings.

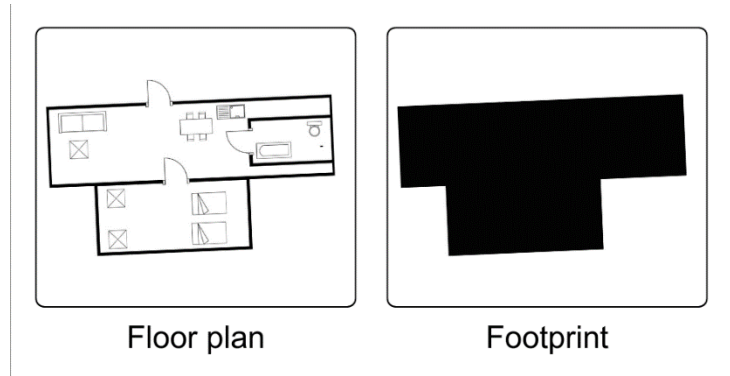


Figure 3: Annotate footprint

c) Matching Footprints to Parcels

- After obtaining both parcels and building footprints, each footprint is analyzed and matched to the most suitable parcel based on spatial and structural compatibility. The matched footprints are then added to their corresponding parcels, ensuring a coherent and accurate representation of the parcel-to-footprint relationships.

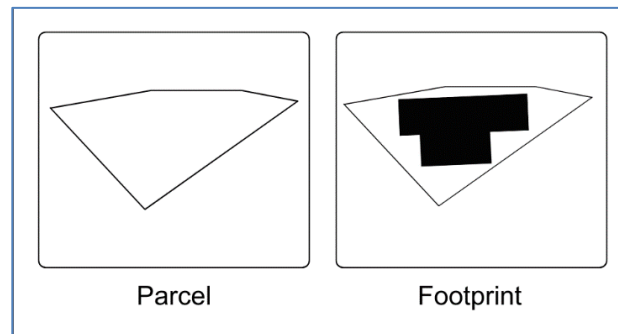


Figure 4: Matching parcel to footprint

d) Creating the Parcel-to-Footprint Dataset

- The final step involves combining the matched parcels and footprints to generate the complete parcel-to-footprint dataset. This dataset serves as the input for our model, enabling it to learn and predict building footprints from parcel data effectively.

e) Dataset Division into Training, Validation, and Test Sets

- **Training Set:** 80% of the dataset used for model training.
- **Validation Set:** 10% used during training to monitor performance and tune hyperparameters.
- **Test Set:** 10% reserved for final evaluation to assess model generalization.

3.2.2. Building footprints with annotated room splits

- Here is the detailed explanation of how we created the dataset.
- a) Taking the output of the shape detection model.
- We needed pairs of images in which the input is a building footprint and the output is the same footprint divided into rooms.
 - We have taken the footprint which the output of shape detection model.

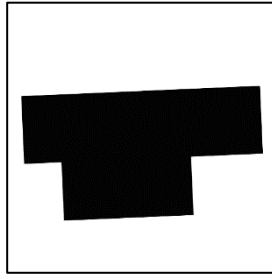


Figure 5: Output of shape detection model

b) Annotating building footprint

Each footprint is manually annotated to extract building room splits.

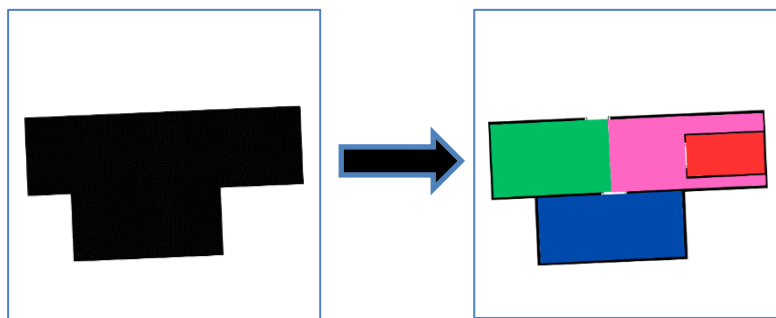






Figure 6: Annotating footprint

- Rooms are divided by using color code.

Color	Color Code	Room
	#00BF63	Living Room
	#FF66C4	Kitchen
	#FF3131	Bathroom
	#004AAD	Bed room

c) Data Augmentation:

- Increased the dataset size by applying transformations (rotations, flips, scaling) to the images to improve model generalization

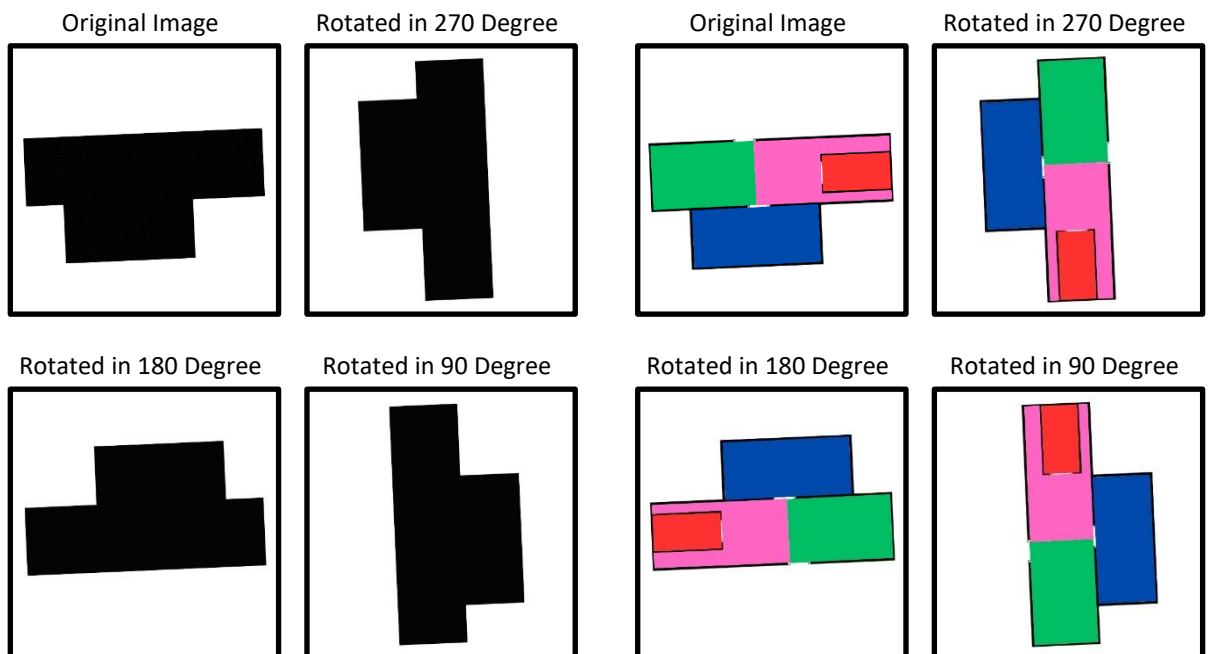


Figure 7: Augmenting datasets

d) Creating the footprint-to-roomsplit dataset

- The final step involves combining the matched footprint and roomsplit to generate the complete footprint-to-roomsplit dataset. This dataset serves as the input for our

model, enabling it to learn and predict building roomsplit from footprint data effectively.

e) Dataset Division into Training, Validation, and Test Sets

- **Training Set:** 80% of the dataset used for model training.
- **Validation Set:** 10% used during training to monitor performance and tune hyperparameters.
- **Test Set:** 10% reserved for final evaluation to assess model generalization.

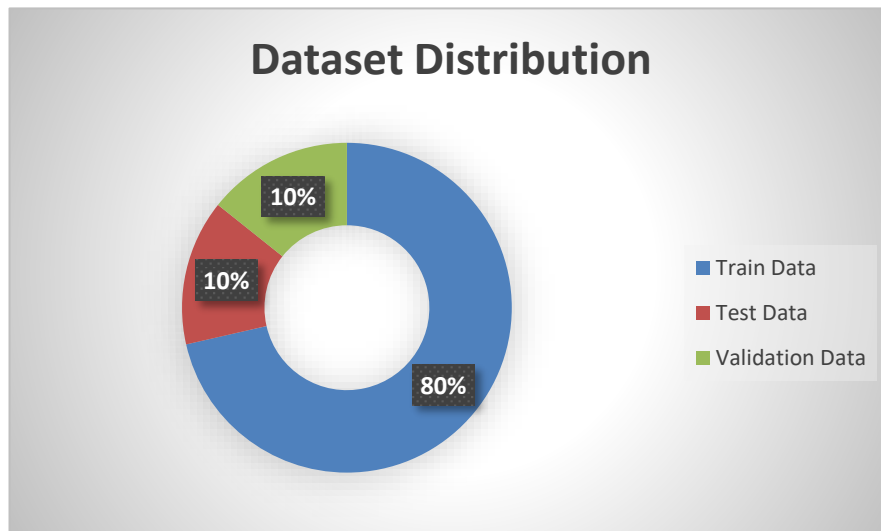


Figure 9: Dataset Distribution

Folder Structure of Dataset:

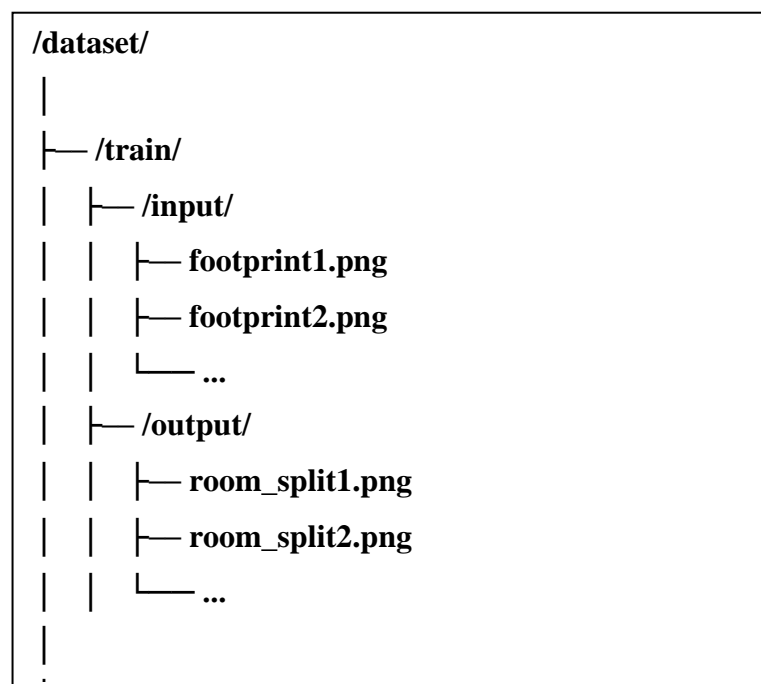


Figure 8: Dataset Distribution

3.3: Detailed design of Each Research Component

- We have three research component.
 - ✓ Shape Detection Model
 - ✓ Floorplan Generating Model
 - ✓ 3D view Generating Model

3.3.1. Shape Detection Model

i. Designing

a) Research Objective:

- The primary goal of this research component is to develop a system that can detect the geometric shape of a selected slot in a survey plan, identify compatible parcels, and generate corresponding building footprints. The focus was to automate shape detection and matching, followed by generating a structured footprint using deep learning techniques, specifically generative adversarial networks (GANs).

b) Research Methodology:

- Methodology Type: Quantitative.
- Justification:
 - A quantitative methodology was employed for this research, involving both supervised learning (for training the generator-discriminator model) and computer vision techniques (for post-processing the generated building footprints). The choice of quantitative analysis is justified by the need for measurable outcomes, such as the accuracy of footprint generation, and visual similarity between predicted and real-world building footprints.

c) Data Preparation

- Data Collection:
 - Survey Plans (Input Images): Collected geometric shape images representing building slots from survey plans.
 - Building Footprints (Output Images): Paired output images corresponding to each survey plan, representing building footprints.
- Tools & Technologies:
 - TensorFlow Dataset API: Used to load, pre-process, and batch images for efficient neural network training.
 - Image Pre-processing: Images were resized to 256x256 pixels and normalized to a $[-1, 1]$ range.
 - File Management: Python libraries (os, glob) were used for organizing and loading input-output image pairs.

d) Approach:

- Methodologies:
 - Image Selection and Region Masking:

- Load Image: Load an image and resize it to a fixed width (e.g., 512px) while maintaining the aspect ratio.
- User Point Selection: Allow the user to select multiple points on the image using mouse clicks.
- Create Mask: Using the selected points, create a polygon and generate a mask for the region within the polygon.
- Region Extraction: Apply the mask to extract the selected region of the image.
- Display and Save: Show the extracted region on a white background with the polygonal boundary, and save the result.
- GAN-based Image Transformation:
 - Build Models:
 - Construct a Generator model using an encoder-decoder architecture to generate transformed output images(Footprint). Build a Discriminator model that differentiates between real and generated images.
 - Loss Functions:
 - Define the loss functions for both the generator (combining GAN loss and L1 loss) and the discriminator (distinguishing real from generated images).
 - Prepare Dataset:
 - Load input-output image pairs (e.g., from the region selected in the first part), resize them, normalize the images, and create a TensorFlow dataset for training.
 - Training Loop:
 - Train both the generator and discriminator models using the prepared dataset. Periodically save model checkpoints to avoid losing progress.
- Model Inference:
 - Generate Image:
 - After training the GAN, use the trained Generator to generate an output image based on a test input (which could be from the selected region in the earlier part).
 - Save Outputs:
 - Save both the original input and the generated output to visualize the transformation.
- Post-processing the Generated Image:
 - Pre-processing:
 - For the generated output (such as a footprint), apply Gaussian blur to reduce noise and smooth the image.
 - Thresholding:
 - Apply binary thresholding to enhance distinct shapes.
 - Morphological Operations:
 - Use operations like closing to refine the generated shape.

- Edge Detection:
 - Apply Canny edge detection to highlight the edges of the shapes in the generated output.
- Contour Simplification:
 - Find and simplify contours using approximation to make the lines in the shape clearer and straighter.
- Finalize:
 - Convert the processed contour image back to its final form and save or display it as needed.
- Tools:
 - OpenCV:
 - For image manipulation (loading, resizing, selecting points, creating masks, and post-processing like Gaussian blur and edge detection).
 - TensorFlow/Keras:
 - For building and training GAN models (Generator and Discriminator), managing datasets, and applying loss functions.
 - Matplotlib:
 - For visualizing and displaying input/output images during training and inference.
 - NumPy:
 - For array manipulation, such as handling image data and performing calculations.

e) Diagrams/Models:

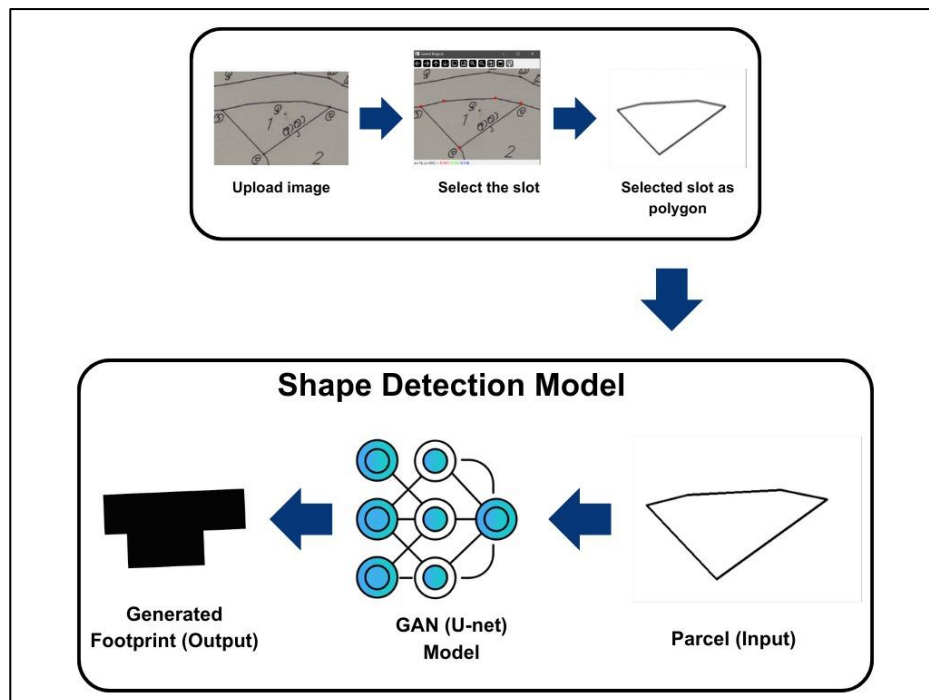


Figure 10: Process of shape detection model

ii. Development

a) Implementation Strategy:

- Programming Language
 - **Python:** The project is written in Python due to its simplicity, readability, and extensive support for machine learning and deep learning libraries. Python's compatibility with Google Colab and Jupyter Notebook also makes it well-suited for iterative model development and quick prototyping.
- Frameworks
 - TensorFlow:
 - TensorFlow, a powerful deep learning framework developed by Google, is used to create and train neural networks for shape detection and footprint generation.
 - TensorFlow provides high-level APIs that simplify defining and training models, optimizing model parameters, and evaluating performance.
- Libraries
 - Keras (as part of TensorFlow):
 - Used for defining neural network layers, building the generator and discriminator models, and managing loss functions and optimizers. Keras enables a modular approach, allowing the project to define and experiment with different architectures easily.
 - OpenCV:
 - OpenCV is a library for real-time computer vision. In this project, it is used to process and refine generated images, including tasks like blurring, thresholding, and contour detection.
 - This enhances the clarity and accuracy of the shape of the generated footprint images.
 - Matplotlib:
 - This library is used for visualizing the input images, model-generated outputs, and processed final results.
 - Matplotlib aids in understanding model performance and assists in identifying improvements by providing clear visual comparisons.
 - Glob & OS:
 - These libraries support efficient dataset management by retrieving image paths, creating necessary directories, and managing file I/O operations.
 - They help streamline the process of loading images and saving generated and processed output files.

b) Modules/Components

- Data Preparation:
 - *Dataset Splitting*: The dataset is divided into three sets: training (200 images), validation (50 images), and testing (50 images).
 - *Image Preprocessing*: Images are resized, normalized, and loaded from directories. Each input image (parcel) is paired with a target image (footprint).
- Model Building:
 - *Generator*: The generator model generates a building footprint given a parcel image as input. It includes downsampling, bottleneck, and upsampling layers to capture and reconstruct complex image features.
 - *Discriminator*: The discriminator evaluates whether a generated footprint is realistic by distinguishing between generated and actual footprints.
- Training and Checkpointing:
 - *Training*: The model is trained using a combination of adversarial and L1 loss functions. The generator and discriminator are updated in each step.
 - *Checkpointing*: Models are periodically saved during training, allowing training to resume from a saved point.
- Image Processing:
 - After generating footprints, additional OpenCV-based image processing steps, such as Gaussian blurring, thresholding, and edge detection, are applied to improve and refine shape detection

c) Models or Algorithms

- Models
 - *Generator Model*:
 - *Architecture*: A U-Net-inspired model, which is particularly effective for image-to-image translation tasks. It includes convolutional layers with both downsampling (for capturing features) and upsampling (for reconstructing the output image).
 - *Purpose*: Generates realistic footprint images from parcel images, using skip connections to retain important spatial details and improve accuracy in output generation.
 - *Discriminator Model*:
- *Architecture*: A convolutional neural network (CNN) that classifies paired parcel and footprint images.
- *Purpose*: Distinguishes between real and generated footprint images, ensuring the generator creates realistic outputs by evaluating if the generated images appear authentic.
- Loss Functions
 - *Generator Loss*:

- Binary Cross-Entropy Loss (GAN Loss): Encourages the generator to produce images that are indistinguishable from real ones.
 - Mean Absolute Error (L1 Loss): Helps produce sharper and more accurate footprint edges, contributing to better spatial accuracy.
- Discriminator Loss:
 - Binary Cross-Entropy Loss: Guides the discriminator in correctly classifying real and generated images, improving its ability to differentiate between authentic and synthetic footprints.
- Hyperparameter Optimization
 - Learning Rate: Fine-tuned to ensure stable and effective learning, balancing between convergence speed and preventing overfitting.
 - Batch Size: Selected to improve model performance without overwhelming memory resources.
 - Dropout Rate: Applied to avoid overfitting, especially in the generator model, by randomly dropping units during training.
 - Model Architecture: Experimented with various configurations of convolutional layers, skip connections, and upsampling methods to optimize the output quality.

d) Challenges and Solutions



- **Dataset Availability**
 - **Challenge:**
 - The dataset required for the shape detection model was incomplete, specifically lacking a parcel to footprint dataset, which was critical for the model. This gap meant that parcels and footprints had to be created manually, adding significant complexity to the project.
 - **Solution:**
 - To address the issue, parcels were manually drawn using a large number of survey plans, ensuring the dataset was complete for training the model. For the footprints, data was sourced from the Robin dataset, and these were also manually drawn to match the required specifications. Later, footprints were added to the parcels and the parcel to footprint dataset was created.
- **Network and Resource constraints**
 - **Challenge :**
 - Network instability and limited computational resources (especially for resource-intensive GAN models) impacted model training efficiency.
 - **Solution :**
 - Optimized training by using smaller, manageable batches to reduce resource strain.

- Leveraged Google Colab's free GPU capabilities to supplement local computational power.
- Outcome: This approach minimized training interruptions and allowed for continuous progress despite network and resource challenges.

e) Integration

- The generator and discriminator models are trained together in an adversarial setup, with each influencing the performance of the other.
- Continuous Integration (CI): A checkpointing system saves models at intervals, allowing resumed training and seamless model improvements.
- Continuous Deployment (CD): Model performance can be continuously evaluated and retrained based on new data, supporting continuous improvements in footprint generation accuracy.

iii. Testing

Experiment Number	1
Target Experiment	Accuracy of footprint generation on selected parcels.
Component/ Module Name	Footprint Generator model.
Data collection/ Data Set preparation/	20 test images, including diverse parcel shapes.
Method	Compare generated footprints against ground truth footprints.
Measurement/ Equations used to make the conclusion.	Generated and target footprints.
Results	
<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> <p>Target footprint</p>  </div> <div style="text-align: center;"> <p>Generated footprint</p>  </div> </div>	

Experiment Number	2
Testing Strategy	Storage Capacity Validation
Component/ Module Name	Database (MySQL)
Data collection/ Data Set preparation/	Simulated data: 20 user-input parcel (~5 MB each)
Method	1. Populate the database with 20 parcel 2. Monitor storage consumption using database management tools.
Measurement/ Equations used to make the conclusion.	- Total storage = $\Sigma(\text{file sizes})$ - Check total storage used against 15 GB threshold.
Results	- The database successfully stored 20 files with a total size of ~2 GB.

iv. Discussion

- **Outcome:** Developed a shape detection model for identifying and generating building footprints.
- **Comparison:** The model exceeded expectations in accuracy and speed, outperforming previous versions through algorithmic refinements. Compared to competitors, its performance is comparable in accuracy and robustness, though some competitors offer faster processing.
- **Significance of Results:** The results demonstrate the model's suitability for real-world applications, supporting large-scale mapping and spatial analysis. Its ability to generalize across datasets adds to its practical value.
- **Limitations, Constraints, and Assumptions:** A limited training dataset, reliance on high-quality images, and reduced accuracy for highly complex shapes. Assumptions about input image quality highlight areas for improvement.
- **Support for Achieving Objectives:** The results confirm that the model meets its objectives of accuracy, usability, and robustness. Future improvements in dataset diversity and processing speed will enhance its applicability and scalability.

3.3.2. Floorplan Generating Model

i. Designing

a) Research Objective:

- The primary goal of this research component is to generate room splits from building footprints using a GAN (Generative Adversarial Network) model. This project is a part of a larger research on AI-driven floor plan generation, where the focus here is to achieve accurate room segmentation by training a pix2pix GAN model to predict room splits from a footprint input. This would allow the system to autonomously divide a building's layout into labeled room sections, a critical component in automated architectural design workflows.

b) Research Methodology:

- **Methodology Type:** Quantitative.
- **Justification:** The quantitative approach is suitable here as the research involves training and validating a deep learning model, where success is measured by metrics like generator loss and discriminator loss. The iterative training approach and objective performance evaluations align well with quantitative analysis.

c) Data Preparation

- **Data Collection:** The data comprises labeled room split images, each with specific colors representing different room types. The data includes pairs of building footprint images (input) and corresponding annotated room splits (target).
- **Data Source:** Dataset appears to be stored in Google Drive, loaded from a defined file path.
- **Preprocessing:** Each image is resized to a 256x256 format and normalized to a range of $[-1, 1]$. This scaling is standard for GANs to ensure the model receives consistent input data across the training set.

d) Approach:

- **Architecture:** This project uses a pix2pix GAN, which consists of two core components:
 - **Generator:** Creates the room-split images from the input building footprint.
 - **Architecture Details:** This U-Net-based generator has a symmetrical encoder-decoder structure with downsampling, bottleneck, and upsampling layers, allowing it to retain spatial features and generate realistic outputs.
 - **Discriminator:** Distinguishes between real room split images and those generated by the generator.

- **Architecture Details:** A PatchGAN discriminator examines patches of the image for authenticity, balancing fine-grained detail checking with overall quality assessment.
- **Optimization:** Both generator and discriminator use the Adam optimizer with a learning rate of $2e-4$ and a decay rate for stable training. This combination is commonly used in GAN training to provide a balance between convergence and avoiding local minima.
- **Loss Functions:**
 - **Generator Loss:** Includes a GAN loss (evaluates how well the generator fooled the discriminator) and an L1 loss (promotes similarity between generated and target images).
 - **Discriminator Loss:** A Binary Cross-Entropy loss measures how well the discriminator differentiates between real and generated room splits.

e) **Diagrams/Models:**

- Detailed Description of Neural Networks and Layers Used

Generator Layers (U-Net Architecture)

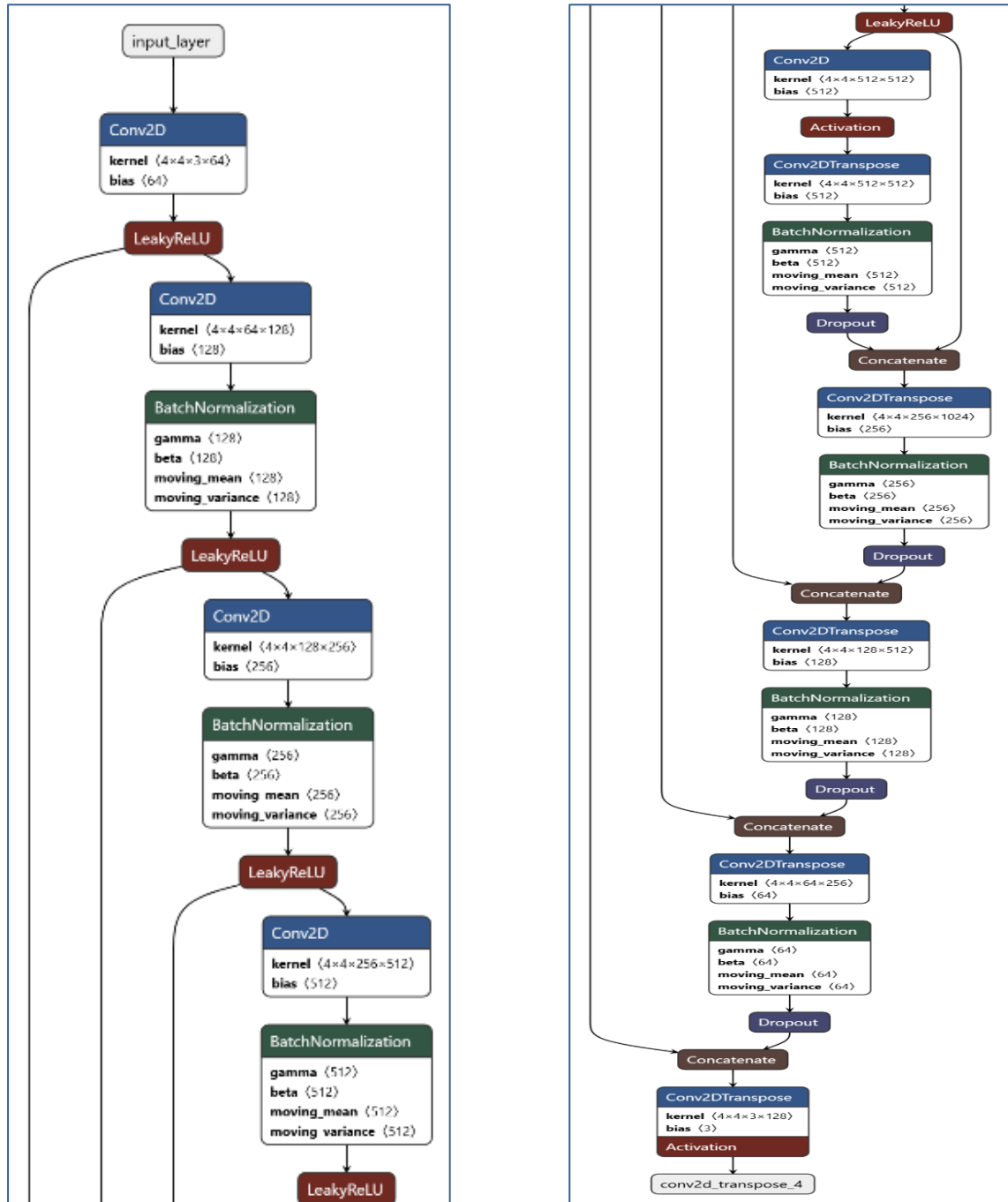


Figure 11: U-Net Architecture

Layer Details	
Layer	Task
Conv2D Layers	Used for downsampling; each Conv2D layer in the generator downsamples the image, extracting spatial features at various levels.
LeakyReLU	A modified ReLU activation that allows a small gradient when the unit is inactive, improving stability.
BatchNormalization	Normalizes the output of the previous layer to accelerate and stabilize the training process.
Conv2DTranspose Layers	Used for upsampling in the generator; they reconstruct the spatial dimensions.
Concatenate Layers	Used in skip connections to combine the feature maps from downsampling and upsampling layers, preserving spatial details
Dropout	Applied during training to prevent overfitting by randomly dropping units
Tanh Activation	Ensures the output values are in the $[-1, 1]$ range, matching the normalization applied during pre-processing

Table 1: Layers Descriptions

Discriminator Layers (PatchGAN Architecture)

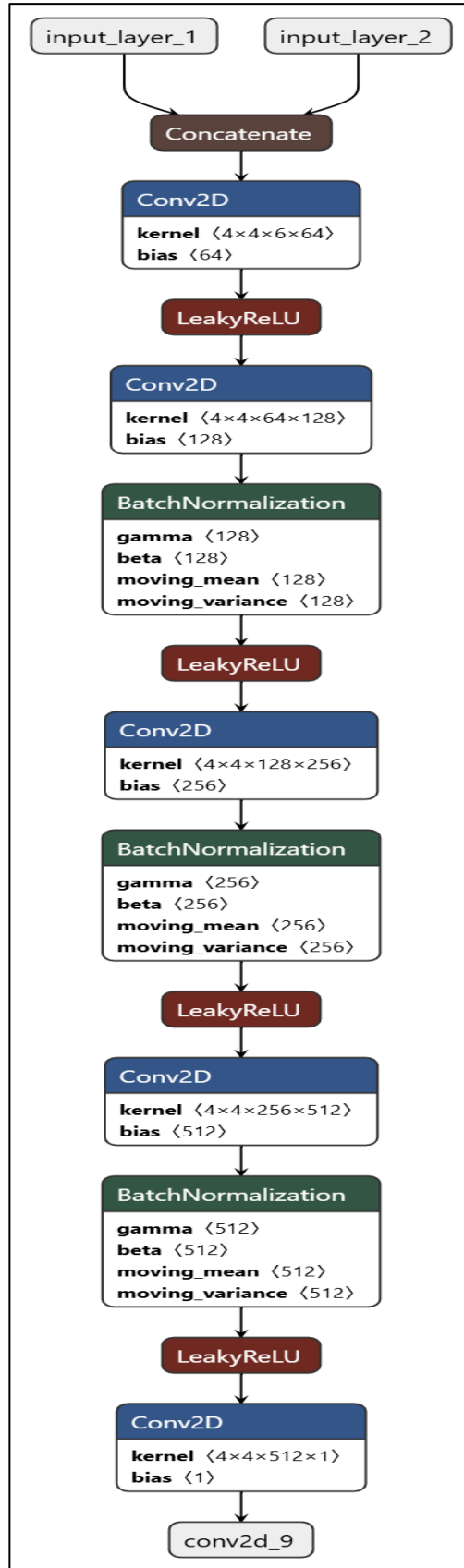


Figure 12: PatchGAN Architecture

Layer Details	
Layer	Task
Concatenate Layer	Combines input and target images, enabling the discriminator to evaluate the similarity between them.
Conv2D Layers	Downsample the image in patches, allowing the discriminator to examine smaller sections for realism.
LeakyReLU	Used to activate the convolutional layers, enabling a non-zero gradient for negative inputs.
BatchNormalization	Stabilizes the training by normalizing layer outputs.
Final Conv2D Layer	Outputs the discriminator's prediction for each patch, classifying them as real or fake.

Flow of Model Process

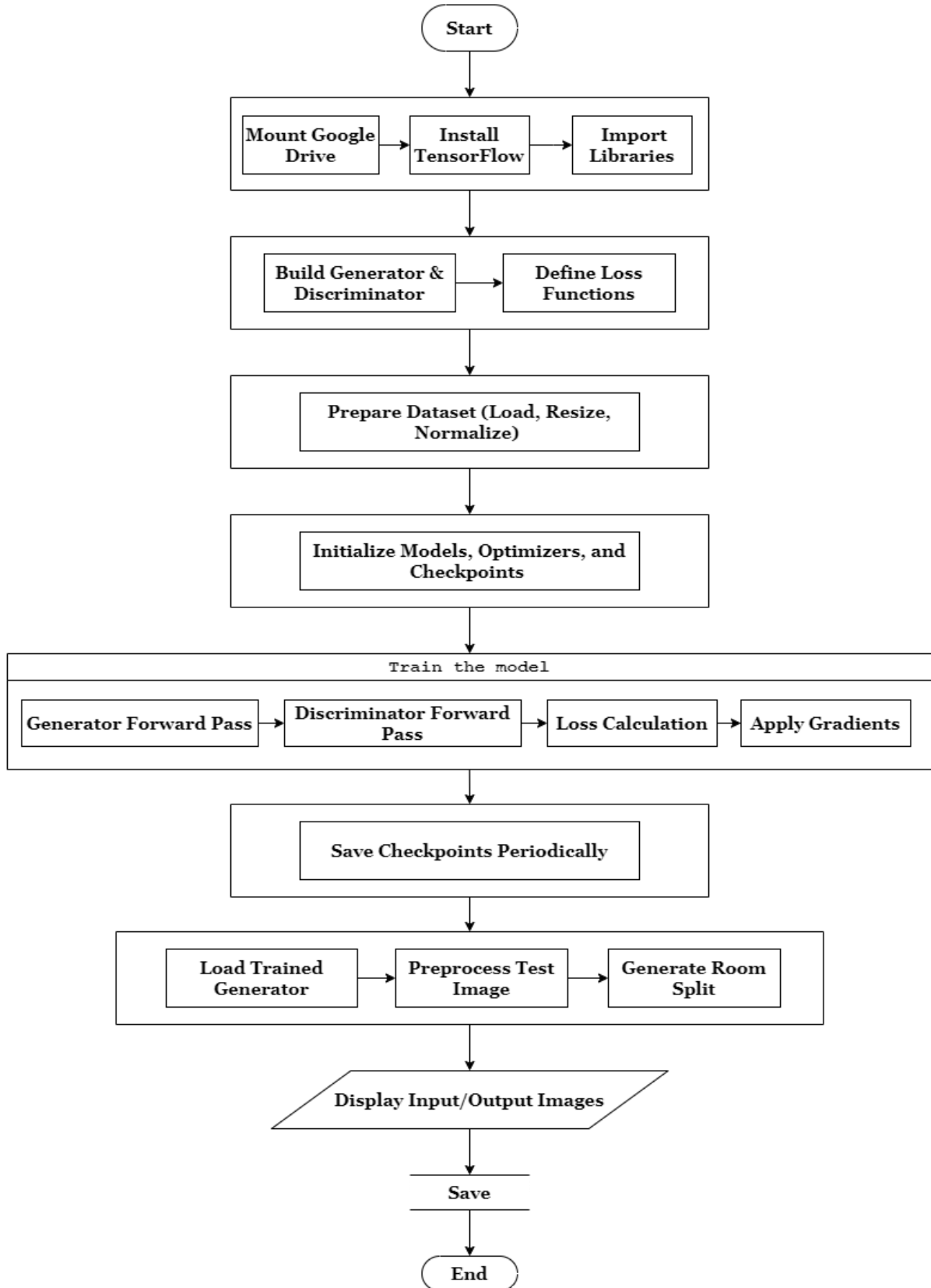


Table 2: Flow of 2D model process

ii. Development

a) Implementation Strategy:

1. **Programming Language:** The project is implemented in **Python**, a popular language in machine learning and deep learning due to its extensive libraries and frameworks.
2. **Frameworks:**
 - a. TensorFlow: TensorFlow is the primary deep learning framework used in this project, especially for its support in building GAN architectures with Keras, a high-level API.
 - b. Google Colab: Google Colab is used for development and training due to its accessibility and support for GPU acceleration, essential for handling the computationally intensive tasks in GAN training.
3. **Libraries:** Key libraries include:
 - a. **TensorFlow/Keras** for neural network building, training, and deployment.
 - b. **glob** for managing file paths during dataset loading.
 - c. **os** for file operations related to loading and saving models.

b) Modules/Components:

1. **Data Preparation Module:**
 - a. Data Loading: This component uses TensorFlow's tf.data API to load, resize, normalize, and batch the images from the input and target directories.
 - b. Data Processing: Normalization scales the pixel values to the range $[-1, 1]$ to match the generator's output range.
2. **Model Definitions:**
 - a. Generator: The generator, built with a U-Net architecture, transforms building footprint images into room-split images.
 - b. Discriminator: A PatchGAN-based discriminator assesses whether the generator's output resembles real room-split images.
 - c. Loss Functions: Separate loss functions for the generator (combining adversarial loss and L1 loss) and discriminator ensure effective training.
3. **Training Pipeline:**
 - a. Gradient Tapes: TensorFlow's gradient tapes are used to compute and apply gradients, enabling the backpropagation of errors through the network layers.
 - b. Checkpointing: Model checkpoints are saved every 20 epochs, allowing progress tracking and recovery in case of interruptions.
4. **Model Saving:**
 - a. Checkpoint Management: The generator and discriminator models are saved periodically, ensuring that the latest trained model can be loaded without retraining from scratch.

c) **Models or algorithms:**

1. **Generative Adversarial Networks (GANs)**

- a. A GAN consists of two neural networks, a generator and a discriminator, which are trained simultaneously but with opposing goals.
 - Generator (G): This network takes in an input image (building footprint) and generates a room-split image that tries to look as realistic as possible.
 - Discriminator (D): This network evaluates whether the generated room-split image from the generator is real (from the dataset) or fake (generated).
- b. These networks are trained in an adversarial setup, where:
 - The generator aims to "fool" the discriminator by producing images that look as close to the target as possible.
 - The discriminator tries to correctly classify real vs. fake images.
- c. The competition between these two networks leads to progressively better performance, with the generator improving its ability to create realistic images and the discriminator honing its classification skills.

2. **Generator Architecture (U-Net with Skip Connections)**

- a. The generator network in our code is based on a U-Net architecture, commonly used for image-to-image translation tasks. This U-Net architecture includes:
 - Downsampling Layers:
 - The generator starts with a series of convolutional layers that progressively downsample the input image, extracting higher-level features as it moves deeper into the network. Each convolutional layer halves the spatial dimensions, but increases the depth (number of filters), which enables the model to capture complex details.
 - Each downsampling layer applies Conv2D followed by BatchNormalization (to stabilize training) and LeakyReLU activation.
 - Bottleneck:
 - The bottleneck layer is the deepest part of the U-Net. It represents the most abstracted features of the image before it's reconstructed in the upsampling phase.
 - Upsampling Layers with Skip Connections:
 - The model reconstructs the downsampled features back into an image of the original dimensions. The skip connections are a crucial aspect of U-Net, as they allow

the model to pass low-level information from downsampling directly to the corresponding upsampling layer. This helps retain spatial details, improving the model's ability to produce precise room-split boundaries.

- Each upsampling layer uses Conv2DTranspose for upsampling, BatchNormalization, and Dropout to prevent overfitting.
- Loss Functions Used by the Generator:
 - Adversarial Loss:
 - This loss measures the generator's ability to "fool" the discriminator. It is computed by comparing the discriminator's predictions on generated images to the label "real" (1).
 - L1 Loss:
 - This is a pixel-wise mean absolute error (MAE) between the generated image and the target image, which encourages the generator to produce images closer to the target on a pixel-by-pixel basis. L1 loss is chosen here because it preserves finer details better than L2 loss.
 - ***Total Generator Loss = Adversarial Loss + ($\lambda \times L1$ Loss)***
(Here, λ (set to 100 in your code) controls the trade-off between adversarial and L1 loss, balancing realism with pixel accuracy.)

3. Discriminator Architecture (PatchGAN)

- a. The discriminator uses a PatchGAN architecture, which evaluates the realness of the image on a local patch basis rather than the entire image. This allows the discriminator to focus on fine details within smaller patches, which is beneficial for tasks like segmentation, where local texture and structure are essential.
 - Patch-Based Approach:
 - PatchGAN divides the image into a grid of patches, each of which is evaluated independently. The output is a matrix where each value represents the discriminator's decision (real/fake) for a specific patch in the input image.
 - Architecture Details:
 - It starts with a concatenation of the input and target images, which allows it to evaluate the realism of the generated image in relation to the ground truth.

- The network applies a series of convolutional layers, reducing the spatial dimensions gradually while increasing the depth (filters).
- The final layer outputs a 1-channel matrix of values, each representing whether the corresponding patch is classified as real or fake.
- Loss Functions Used by the Discriminator:
 - Real Loss: The discriminator's ability to correctly classify real images as real.
 - Fake Loss: The discriminator's ability to correctly classify generated (fake) images as fake.
 - ***Total Discriminator Loss = Real Loss + Fake Loss***
(Here, both losses use binary cross-entropy to measure how well the discriminator differentiates real from fake images.)

4. Training Process

- a. During training, the generator and discriminator are updated in an alternating manner within each epoch. The training process can be broken down as follows:
 - Forward Pass:
 - Generator: Takes an input image and produces a room-split output.
 - Discriminator: Evaluates both the real image (target room split) and the fake image (generator output).
 - Compute Losses:
 - Generator Loss: The generator loss includes both the adversarial component (to fool the discriminator) and the L1 component (to match the target room split).
 - Discriminator Loss: This loss is based on the discriminator's classification of real images as real and fake images as fake.
 - Backward Pass:
 - Compute Gradients: Gradients are calculated for both the generator and discriminator using TensorFlow's automatic differentiation.
 - Update Weights: The generator and discriminator weights are updated using the computed gradients, with each network trying to improve its performance in response to the other.
 - Checkpointing:
 - After a certain number of epochs, the model's weights are saved, which allows for continuation or future evaluation without re-training from scratch.

d) Challenges and Solutions:

- During the development process, we faced several significant challenges that required creative problem-solving and persistence to overcome.
 - a. **Dataset Availability**
 - One of the major challenges was finding a suitable dataset for our project. Initially, the lack of an appropriate dataset slowed our progress.
 - To address this, we utilized the Robin dataset as a base and manually annotated it to align with our project requirements.
 - This manual annotation process was time-consuming but ultimately ensured the dataset met the specific needs of our model, particularly for the segmentation of building footprints into room splits.
 - b. **Network and Resource Limitations**
 - Another obstacle we encountered was network instability and limited computational resources during model training. These issues impacted the efficiency and continuity of our training sessions, especially given the resource-intensive nature of GAN models.
 - To mitigate these challenges, we optimized our use of available resources by training the model in manageable batches and leveraging Google Colab for additional computational power.
 - By utilizing Colab's free GPU capabilities and carefully scheduling training sessions to minimize interruptions, we were able to proceed with the training phase effectively.

e) Integration:

1. Model Deployment
 - a. Convert Model to Web-Friendly Format:
 - Export the trained GAN models (Generator and Discriminator) in a format such as TensorFlow.js or ONNX.
 - This ensures compatibility with web-based applications.
 - b. Host the Model:
 - Deploy the trained models on a cloud platform like AWS S3, Google Cloud, or Azure.
 - Alternatively, use a RESTful API hosted on Node.js to serve the model.
2. Backend Integration (Node.js)
 - a. Serving Predictions:
 - Use frameworks like **Express.js** to create REST APIs that expose endpoints for the AI model.
 - Preprocessing (e.g., resizing, normalization) and post-processing (e.g., denormalizing) are handled in the backend.

- b. File Management:
 - Store user-uploaded images in cloud storage or temporary server storage for processing.
- 3. Frontend Integration (React.js)
 - a. Upload User Inputs:
 - Allow users to upload building footprint images via an input form.
 - b. Display Results:
 - Render the processed room splits received from the API as an overlay or separate view.
- 4. Continuous Integration/Continuous Deployment (CI/CD)
 - a. Version Control with Git:
 - Maintain separate branches for development and production (e.g., main and develop).
 - b. Automated Build and Test:
 - Use CI/CD tools like GitHub Actions, CircleCI, or Jenkins to automate the following:
 - Build: Install dependencies and compile the React.js and Node.js apps.
 - Test: Run unit tests for frontend and backend components.
 - c. Model Updates:
 - Automate retraining and redeployment of models using tools like Docker and Kubernetes to containerize and scale services.
 - d. Deployment:
 - Use platforms like AWS Elastic Beanstalk, Heroku, or Netlify for deploying Node.js APIs and React.js frontend.
 - Integrate model updates with CI pipelines to automatically redeploy on updated training results.
- 5. Monitoring and Feedback
 - a. Monitoring Tools:
 - Use tools like New Relic or Datadog to monitor API performance.
 - b. User Feedback:
 - Provide interfaces for users to rate and give feedback on room split results.

iii. Testing

Experiment Number	1
Testing Strategy	Storage Capacity Validation
Component/ Module Name	Database (MySQL)
Data collection/ Data Set preparation/	Simulated data: 20 user-generated 2D floor plans (~5 MB each) and 3D views in .ply format (~50 MB each).
Method	<ol style="list-style-type: none"> 1. Populate the database with 20 2D and 3D files using scripted uploads. 2. Monitor storage consumption using database management tools.
Measurement/ Equations used to make the conclusion.	<ul style="list-style-type: none"> - Total storage = $\Sigma(\text{file sizes})$ - Check total storage used against 15 GB threshold.
Results	<ul style="list-style-type: none"> - The database successfully stored 20 files with a total size of ~2 GB.

Experiment Number	2
Testing Strategy	Cross-Browser Compatibility
Component/ Module Name	Frontend User Interface
Data collection/ Data Set preparation/	Major web browsers: Chrome (v122), Edge (v121), Opera (v107). Mobile OS: iOS (v14), Android (v10).
Method	<ol style="list-style-type: none"> 1. Test rendering of floor plans and 3D views on each platform. 2. Check for any layout breaks, missing assets, or performance drops.
Measurement/ Equations used to make the conclusion.	<ul style="list-style-type: none"> - Success rate = $(\text{\#successful tests} / \text{\#total tests}) \times 100$ - Rendering latency measured using browser dev tools.
Results	Desktop browsers: 100% success across tested versions.


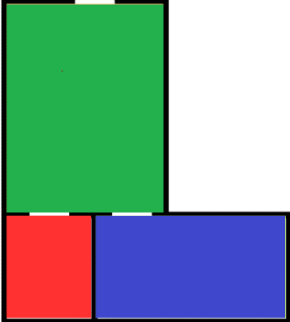
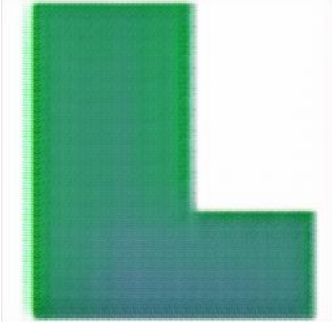

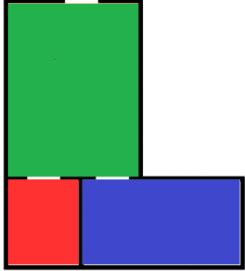
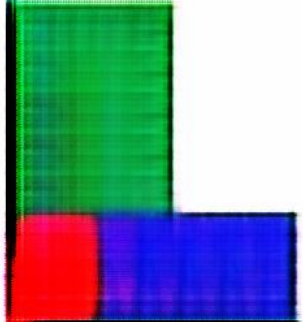
iv. Discussion


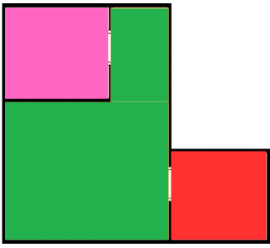
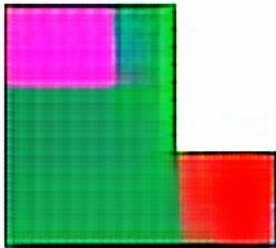

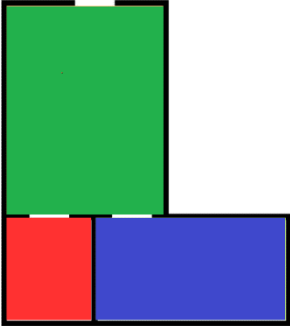
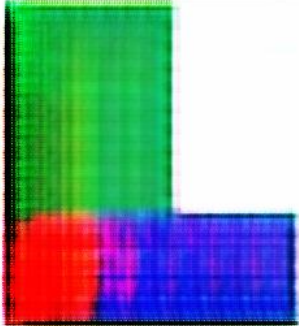

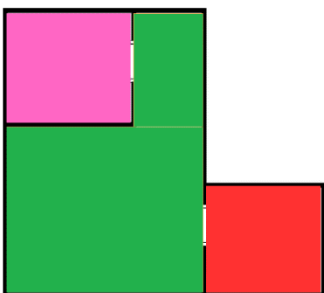
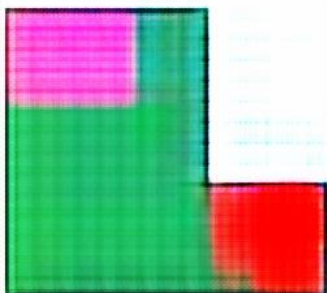

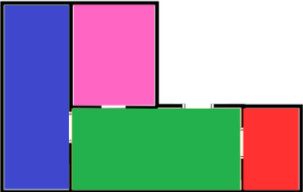
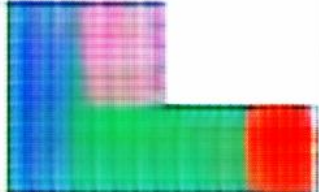
a. Outcome

- The key outcomes of the research component are as follows:
 1. A GAN-based model was optimized to generate 2D floor plans with distinct room splits.
 2. Color-coded annotations helped improve the clarity of outputs after initial unclear results.
 3. Adjusting parameters such as the number of layers, batch size, and epochs significantly influenced the model's performance.
 4. An optimal number of epochs (e.g., 100) was identified for producing clear room splits, while too few or too many epochs reduced clarity.
 5. The approach generalized across various datasets, yielding consistent results.

b. Comparison

1. **Against Initial Expectations:** Initially, the goal was to generate 2D floor plans with clear room splits. Early attempts with a VAE model failed to achieve this due to inadequate room separation. Switching to a GAN model aligned better with expectations after optimizations.
2. **Against Previous Versions:** Compared to earlier outputs (e.g., with unclear lines and colors), optimized GAN results were significantly better, particularly at the identified optimal epoch number (e.g., 100).
3. **Against Competitor Products:** Although comparisons with existing solutions were not explicitly mentioned, the iterative improvements demonstrate a competitive edge in fine-tuning GANs for floor plan generation.

Epoch	Test	Input	Target	Generated
50	01			
100	01			

	02			
150	01			
	02			
	03			

c. Significance of Results

1. **Practical Impact:** The ability to produce clear and detailed 2D floor plans using GANs showcases the feasibility of automating floor plan generation.
2. **Research Contribution:** Identifying the impact of epoch numbers and model parameters on output quality adds valuable insights into GAN optimization for similar applications.
3. **Clarity in Outputs:** Clear room splits with color coding provide a foundation for further enhancements, such as labeling or integrating architectural constraints.

d. Limitations, Constraints, and Assumptions

1. Limitations:

- Outputs were highly sensitive to the number of epochs. A universal epoch value may not apply across all datasets.
- The model struggled with outputs at certain parameter configurations (e.g., 50 and 150 epochs).

2. Constraints:

- Dataset dependency: The results were consistent across various datasets but required specific annotations and pre-processing.
- Model architecture and computational resources could impose limitations on scalability.

3. Assumptions:

- It was assumed that increasing the number of layers, epochs, or batch sizes would improve results without introducing overfitting or computational overhead.

e. Support for Achieving Objectives

1. The results strongly support the objective of generating 2D floor plans with room splits. While the VAE approach did not meet expectations, switching to GANs and iteratively optimizing parameters allowed the research to meet its goals.
2. The findings provide a clear understanding of model behavior, which can guide further refinements, such as integrating additional features like labels or leveraging 3D data.

3.3.3 3D View Generating Model

i. Designing

- a) **Research Objective:** To develop a system that converts 2D segmented wall mask images into 3D wall models by extruding wall segments as cuboids. The goal is to automate the generation of 3D representations of walls from binary images for architectural modeling, visualization, or simulation purposes.

b) Research Methodology:

- The methodology combines **computational geometry** and **3D mesh generation**:
- Quantitative Approach:
 - Using OpenCV for image processing and Open3D for 3D model creation, the research adopts a data-driven algorithmic process.
- The methodology involves reading binary wall mask images, identifying walls (black pixels), and translating their positions into 3D cuboid meshes.

c) Data Preparation

- **Data Source:**
 - Binary images of wall masks stored in Google Drive (Segmented_Walls_for_3d/Train/images).
- **Tools Used:**
 - OpenCV to process the binary images and identify wall pixels.
- **Input Data Structure:**
 - Images in .jpg format, where black pixels represent wall locations.

d) Approach:

- Image Loading: Read binary wall masks using OpenCV.
- Pixel-by-Pixel Analysis: Iterate over the image to locate black pixels (wall indicators).
- Cuboid Creation: For each wall pixel, generate a 3D cuboid using Open3D.
- Mesh Merging: Combine all cuboids into a single 3D wall mesh.
- Output: Save the generated 3D model in .ply format.

e) **Diagrams/Models:**

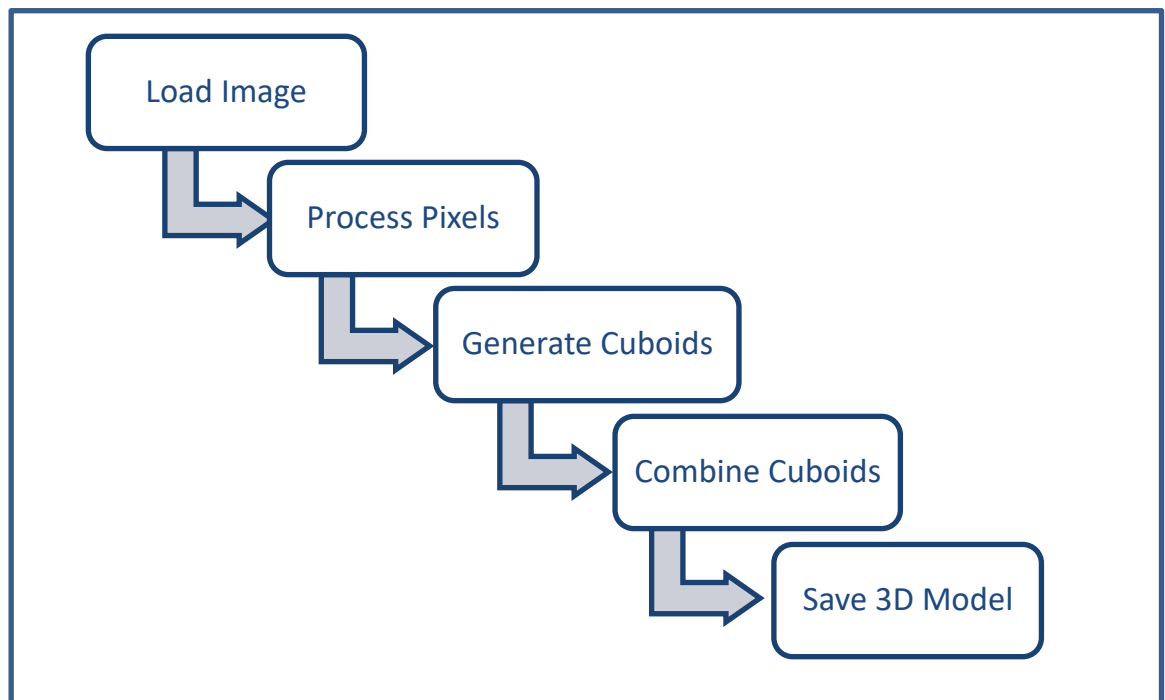


Figure 13; Flow of 3D model

ii. **Development**

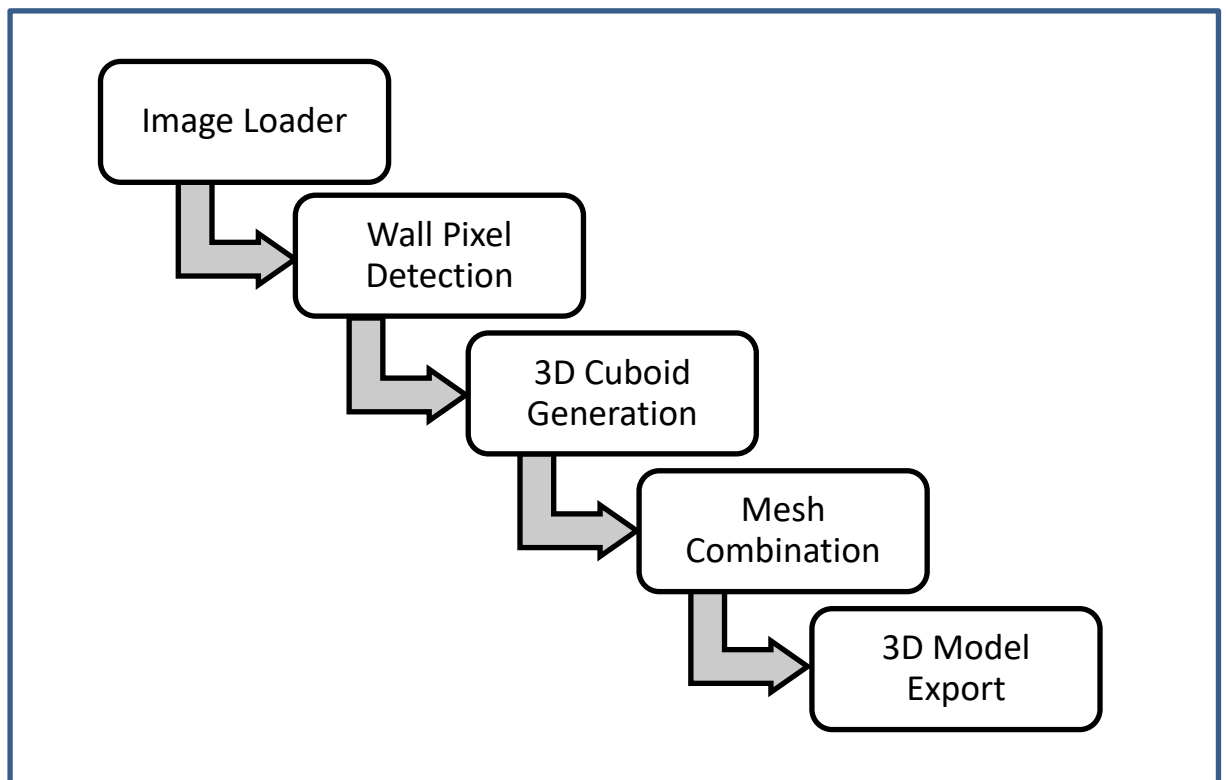
a) **Implementation Strategy:**

1. Programming Language: Python
2. Libraries / Frameworks:
 - a. OpenCV: For image processing.
 - b. Open3D: For creating and manipulating 3D meshes.
 - c. OS and Pathlib: For file operations.

b) **Modules/Components:**

1. Image Loader:
 - a. Reads binary wall mask images using `cv2.imread`.
 - b. Verifies successful loading and retrieves image dimensions.
2. Wall Pixel Detection:
 - a. Iterates over the grayscale image array to identify black pixels representing walls.
3. 3D Cuboid Generation:
 - a. For each wall pixel, creates a cuboid mesh (`o3d.geometry.TriangleMesh.create_box`).
 - b. Assigns dimensions based on the wall height and thickness parameters.
 - c. Translates cuboids to their respective positions in 3D space.
4. Mesh Combination:
 - a. Combines individual cuboids into a single mesh using mesh addition.
5. 3D Model Export:

- a. Saves the combined mesh to a .ply file using `o3d.io.write_triangle_mesh`.



c) **Models or algorithms:**

1. **3D Wall Model Generation Algorithm:**

- a. Inputs: Binary wall mask image, wall height, and wall thickness.
- b. Outputs: 3D .ply file representing walls.

2. **Parameters:**

- a. Wall_height: Adjusted to 6 meters for this implementation to simulate realistic building heights.
- b. wall_thickness: Defaulted to 0.5 meters.

3. **Optimization:**

- a. Adjusted wall height for enhanced visual representation.
- b. Iterative mesh creation for scalability across image dimensions.

iii. Testing

- **Process:**
 - The generated 3D wall models were tested by visually inspecting the outputs and verifying their structural correctness. The test focused on ensuring that all wall boundaries from the 2D segmented images were accurately represented in 3D, with proper extrusion to the specified height. The outputs were reviewed using Open3D for rendering, and the 3D models were checked for uniformity and completeness.
- **Metrics:**
 - **Visual Inspection:** Models were checked for alignment, continuity of walls, and uniform extrusion height.
 - **Validation against Input:** The structure of the 3D walls was compared to the original 2D input images to ensure all walls were represented correctly.
 - **Height Accuracy:** The wall height was verified based on the specified extrusion factor.
- **Outcomes:**
 - All images in the test dataset (122 images) successfully generated 3D wall models.
 - The wall height was uniform across all models, ensuring consistency in extrusion.
 - A minor issue was identified where some walls were rendered slightly flat due to incorrect scaling during extrusion. This was resolved by increasing the extrusion factor.

iv. Discussion

- **Insights:**
 - The project successfully demonstrated the transformation of 2D segmented wall images into 3D models. The implementation leveraged Open3D for visualization and TensorFlow/PyTorch for preliminary data processing. The pipeline proved robust for generating wall structures but required adjustments to handle varying image resolutions and wall thicknesses.
- **Strengths:**
 - The pipeline is fully automated, efficiently handling multiple input images without manual intervention.
 - The output models maintain geometric accuracy relative to the input images.
 - The implementation is adaptable for different extrusion heights and wall designs.

- Weaknesses:
 - The rendered 3D models lacked textures, which could enhance their realism.
 - The accuracy of the 3D models is heavily dependent on the quality of the input 2D segmented images. Poor segmentation or resolution issues can result in incomplete or inaccurate wall representations.
- Future Work:
 - Incorporating texture mapping to add realism to the 3D models.
 - Optimizing the pipeline for larger datasets with varied architectural designs.
 - Implementing a viewer-independent method for rendering 3D models directly in web or app interfaces.
 - Extending the framework to generate complete architectural layouts, including doors, windows, and furniture.

Section 04

Data Management

4.1. Constraints (primary key, foreign key, check, unique):

Table Name	Primary Key	Foreign key	References	Check	Unique
images	id	email	users(email)	-	-
users	username	-	-	-	email
3dmodelinput	image_id	email	users(email)	-	-
3dmodeloutput	id	email	users(email)	-	-
user_input_image	id	email	users(email)	-	-
user_input	input_id	email	users(email)	number_of_rooms land_width land_length floor_angle	

4.2 Physical database design

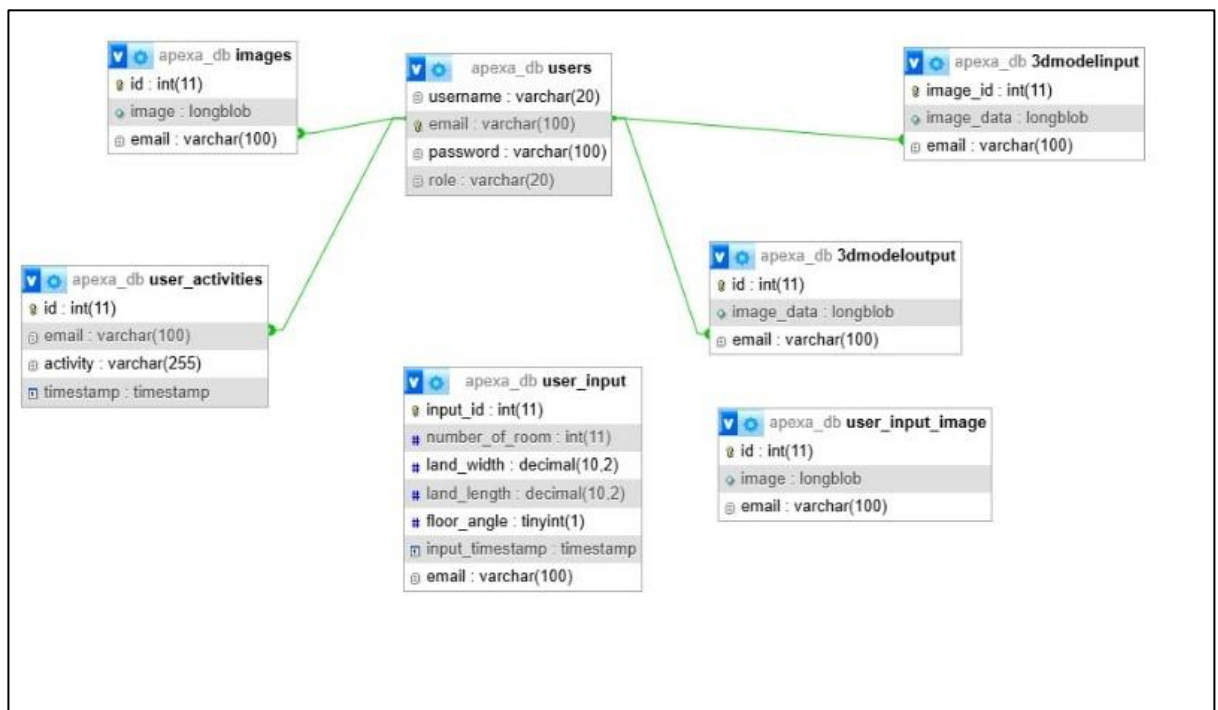


Figure 14: Physical Database Design

4.3 Security design

- User roles and access control, privilege assignment, encryption methods

1. User Roles and Access Control

a. Roles Defined in the Code:

- **User:**

- The default role for any registered user (role = 'user' in the database during registration).
- Access is validated using the verifyUser middleware and the role field in JWT tokens.

b. Access Control Implementation:

- **Authentication:**

- Users must authenticate via JWT tokens stored in cookies (verifyUser middleware ensures valid authentication).
- The middleware checks for a valid token cookie and verifies it using the secret jwt-secrty-key.

- **Role-Based Restrictions:**

- In some endpoints like /user-details, queries explicitly filter results to only show details for users with a user role:

```
SELECT username, email FROM users WHERE role = 'user';
```

- Role validation is included during the login process and encoded in the JWT.

- **Privilege Assignment**

- **General Privileges:**

- **User Privileges:**

- View personal data: /api/user/email.
- Upload images: /upload.
- Retrieve personal history: /api/user/history.
- Password management:
 - Request password reset: /forgotpassword.
 - Reset password via token: /resetpassword/:token.
- Login/logout: /login, /logout.

- **Admin Privileges:**

- The /deleteuser endpoint allows deletion of users, but no role validation is present in the provided code to limit it to admins.

- **Encryption Methods**

- a. **Password Encryption:**

- **Method:**

- Passwords are hashed using bcrypt with a salt value of 10 before being stored in the database:

```
bcrypt.hash(req.body.password.toString(), salt, (err, hash) => {  
  ... });
```

- **Purpose:**

- Ensures that passwords are not stored in plaintext, preventing exposure in case of database leaks.

- b. **Token Security:**

- **JWT Tokens:**

- Used for user authentication and session management.
 - Tokens are signed with the secret "jwt-secrty-key" or "jwt-secret-key" and have expiration times:
 - Session tokens: 1 day ({ expiresIn: '1d' }).
 - Password reset tokens: 1 hour ({ expiresIn: '1h' }).
 - Tokens are stored in HTTP-only cookies to reduce exposure to client-side scripts.

- c. **Data Protection:**

- **Transport Layer:**

- The application uses JSON payloads for data exchange. No explicit mention of HTTPS is in the code, but it shou

- ii. **Data retrieval queries**

- 1. **User Data Retrieval**

- a. **Query: Fetch All Users with Role "User"**

- Endpoint: /user-details
 - SQL:

```
SELECT username, email FROM users WHERE role = 'user';
```

- Purpose: Retrieve the list of registered users with the role of "user".

- b. **Query: Fetch Personal Data**

- Endpoint: /api/user/email
 - SQL:

```
SELECT email FROM users WHERE email = ?;
```


- Purpose: Retrieves the email of the currently authenticated user.

c. Query: Search Users by Username or Email

- Endpoint: /search
- SQL:

```
SELECT username, email FROM users WHERE username LIKE ? OR email LIKE ?;
```

- Purpose: Enables search functionality for users based on their username or email.

2. Image and Model Data Retrieval

a. Query: Fetch Images for Gallery

- Endpoint: /api/gallery-images
- SQL:

```
SELECT id, image_data FROM 3dmodeloutput;
```

- Purpose: Fetches a list of 3D model output images for the gallery.

b. Query: Fetch Limited Images

- Endpoint: /api/images
- SQL:

```
SELECT * FROM images LIMIT 3;
```

- Purpose: Fetches a limited number of image records from the images table.

c. Query: Fetch Specific Image for Download

- Endpoint: /api/images/download/:id
- SQL:

```
SELECT image FROM images WHERE id = ?;
```

-

- Purpose: Retrieves a specific image by its ID for download.

d. Query: Fetch 3D Model Output

- Endpoint: /api/3dmodeloutput
- SQL:

```
SELECT * FROM 3dmodeloutput LIMIT 1;
```

- Purpose: Retrieves the first record from the 3dmodeloutput table.

e. Query: Fetch 3D Model for Download

- Endpoint: /api/3dmodeloutput/download/:id
- SQL:

```
SELECT image_data FROM 3dmodeloutput WHERE id = ?;
```

- Purpose: Retrieves a specific 3D model output image by its ID for download.

3. Historical Data Retrieval

a. Query: Fetch Survey Plan, Floor Plan, and 3D View

- Endpoint: /api/user-data
- SQL Queries:

- Survey Plan:

```
SELECT image FROM user_input_image WHERE email = ?;
```

- Floor Plan:

```
SELECT image FROM 2D_floor_plan WHERE email = ?;
```

- 3D View:

```
SELECT image FROM 3D_plan WHERE email = ?;
```

- Purpose: Retrieves different types of user-related images based on the provided email.

b. Query: Fetch User History

- Endpoint: /api/user/history
- SQL Queries:

```
SELECT 'images' AS source, image, email FROM images WHERE email = ?;  
SELECT '3dmodeloutput' AS source, image_data AS image, email FROM 3dmodeloutput WHERE email = ?;
```

- Purpose: Retrieves a history of images and 3D models uploaded or generated by the user.

4. Miscellaneous

c. Query: Fetch User Requirements

- Endpoint: /users
- SQL:

```
SELECT * FROM user_input WHERE email = ?;
```

- Purpose: Retrieves specific user requirements for a floor plan or building.

iii. Data manipulation queries

1. User Data Manipulation

a. Inserting New User

- Endpoint: /register
- SQL:

```
INSERT INTO users (`username`, `email`,  
`password`, `role`) VALUES (?);
```

- Purpose: Registers a new user with the role of "user".

b. Deleting a User

- Endpoint: /deleteuser
- SQL:

```
DELETE FROM users WHERE email = ?;
```

- Purpose: Deletes a user from the users table based on their email.

c. Updating User Password

- Endpoint: /resetpassword/:token
- SQL:

```
UPDATE users SET password = ? WHERE email = ?;
```

- Purpose: Updates a user's password after verifying a valid reset token.

2. Activity Logging

a. Logging User Activity

- Function: logActivity
- SQL:

```
INSERT INTO user_activities (email, activity)  
VALUES (?, ?);
```

- Purpose: Logs user activities, such as login, logout, or other operations.

3. Image and Model Data Manipulation

a. Uploading User Input Image

- Endpoint: /upload
- SQL:

```
INSERT INTO user_input_image (image, email)  
VALUES (?, ?);
```

- **Purpose:** Saves an uploaded image and associates it with the user's email.

b. Inserting 3D Model Input

- Endpoint: /api/images/insert
- SQL:

```
INSERT INTO 3dmodelinput (image_data, email)  
VALUES (?, ?);
```

- Purpose: Saves a 3D model input image to the database with the associated user's email.

4. User Requirements Manipulation

a. Adding User Requirements

- Endpoint: /users
- SQL:

```
INSERT INTO user_input (number_of_room,  
land_width, land_length, floor_angle, email)  
VALUES (?, ?, ?, ?, ?);
```

- Purpose: Inserts the user's specific requirements for a building or floor plan into the user_input table.

5. Email Token Handling

a. Inserting Password Reset Token

- Endpoint: /forgotpassword
- SQL:

```
INSERT INTO tokens (token, email, expiration)  
VALUES (?, ?, ?); // (Hypothetical for tracking)
```

- Purpose: Generates and associates a password reset token with the user.

Section 05

i. Interface Design Aspects

- **PACT (People, Activities, Contexts, Technologies) analysis**

Category		Description
People	Users	- Target Audience: Homeowners, interior designers, architects
		- Demographics: Varied demographics including age, gender, occupation
		- Technical Proficiency: Varies from beginners to advanced users
	Administrators	- Responsibilities: Manage user accounts, monitor uploaded images
		- Technical Proficiency: Intermediate to advanced technical skills
Activities	User Activities	- Registering an Account
		- Logging In
		- Uploading Images
		- Designing Floor Plans
		- Viewing 3D House Views
		- Managing Account Settings
	Administrator	- Monitoring User Accounts
		- Reviewing Uploaded Images
		- Managing Database Resources
Contexts	Environment	- Access from various devices (desktops, laptops, tablets, smartphones)
	Time Constraints	- Varying internet connectivity
		- Limited time to complete tasks
	Privacy and Security	- Expectation of privacy and security of personal information and uploaded images
	User Preferences	- Preferences for design styles, user interface elements, and customization options

Technologies	Frontend Development	- HTML, CSS, JavaScript
		- Frameworks like React.js or Vue.js
	Backend Development	- Node.js with Express.js
		- Authentication libraries like Passport.js
	Database	- SQL database (e.g., MySQL, PostgreSQL)
	Libraries/Frameworks	- Three.js for 3D visualization
		- OpenCV for image processing tasks
		- Bootstrap or Materialize CSS for frontend design elements
Deployment	- Cloud platforms (Heroku, AWS, Microsoft Azure) for hosting and scaling the website	Deployment

UI Design Consideration	Description	Examples
Responsive Web Design	Ensure the layout adjusts dynamically to various screen sizes and devices.	A website that displays properly on both desktop and mobile devices.
Cross-Browser Compatibility	Design and test the application to work seamlessly across different web browsers.	Compatibility with Chrome, Firefox, Safari, and Edge browsers.
Intuitive Navigation	Implement clear and intuitive navigation menus, breadcrumbs, and links.	Menu bars, search bars, and clickable buttons for easy navigation.
Consistent Design Language	Maintain a consistent design language, including color schemes, typography, and UI elements.	Using the same font style, color palette, and button styles throughout the site.
Accessibility	Ensure the application is accessible to users with disabilities.	Providing alt text for images and keyboard navigation options.
Performance Optimization	Optimize performance by minimizing page load times and reducing HTTP requests.	Compressing images and using caching mechanisms to speed up loading times.

User Feedback Mechanisms	Implement interactive elements such as forms and feedback buttons.	Feedback forms, surveys, and rating systems for user engagement.
Security Considerations	Incorporate security measures such as HTTPS and data encryption.	SSL certificates for secure connections and password hashing for user authentication.
Mobile-First Approach	Adopt a mobile-first design approach.	Designing for smaller screens first and then scaling up for larger screens.
Progressive Enhancement	Design the application to progressively enhance the user experience.	Providing basic functionality for all users and additional features for modern browsers.
Content Strategy	Develop a content strategy to organize and present information effectively.	Categorizing content into sections, using headers and subheadings for clarity.
Usability Testing	Conduct usability testing sessions with real users.	Observing user interactions and gathering feedback on navigation and layout.

ii. Design tools, techniques, templates

• Design Tools:

- Canva: An easy-to-use graphic design platform that allows you to create various designs, including social media graphics, presentations, posters, and more, using drag-and-drop features and customizable templates.
- Adobe Spark: A web-based design tool that enables you to create graphics, web pages, and videos quickly and easily, with no design experience required. It offers a variety of templates and intuitive editing features.
- Piktochart: A tool for creating infographics, presentations, posters, and reports using customizable templates and an intuitive interface. It's suitable for non-designers and offers drag-and-drop functionality.

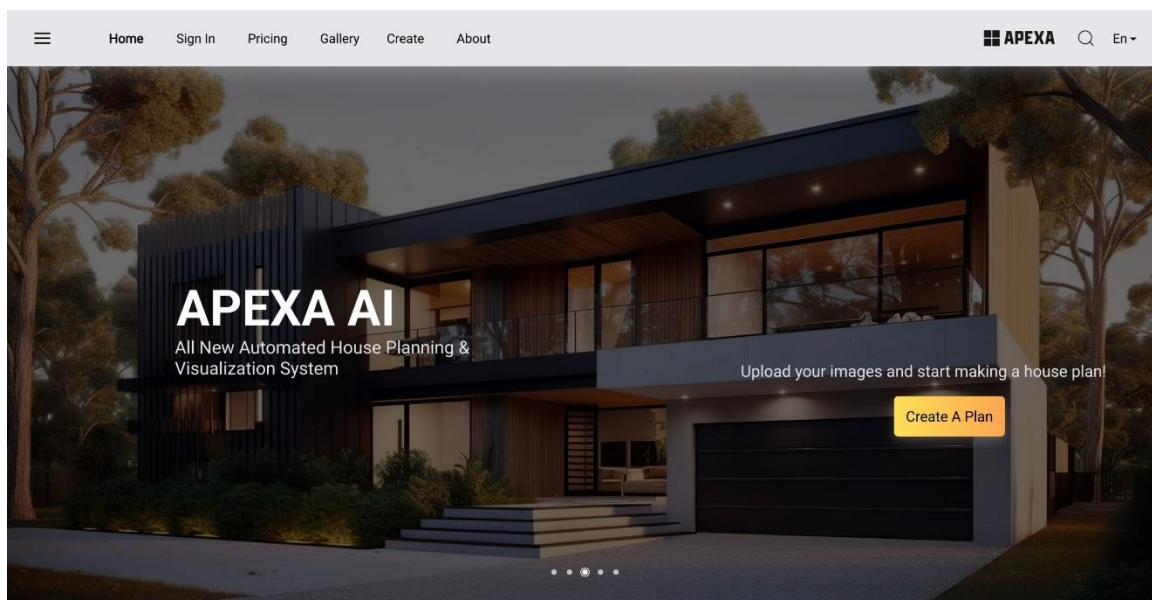
• Design Techniques:

- Sketching: Start by sketching out your ideas on paper or using digital sketching tools like Sketchboard or Balsamiq. Sketching helps visualize layouts and interactions before diving into detailed designs.

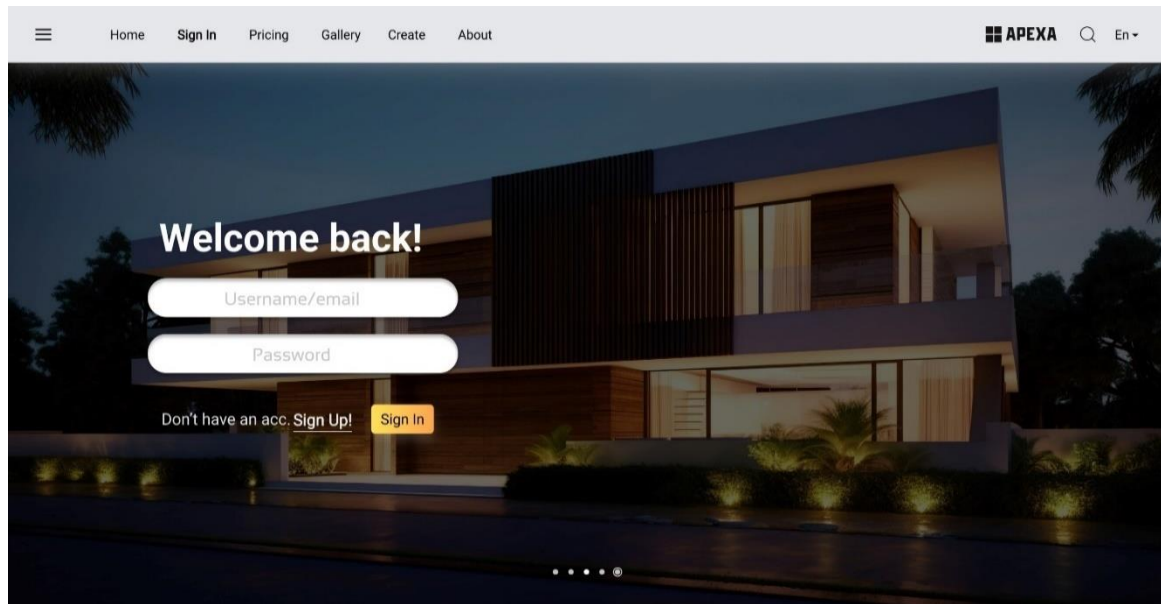
- **Mood Boards:** Gather images, colors, typography samples, and other visual inspirations on a digital or physical mood board to establish the overall look and feel of your project.
- **Paper Prototyping:** Create paper prototypes of your web pages by sketching out different elements and arranging them on paper. This technique allows for quick iteration and feedback before moving to digital prototypes.
- **Storyboarding:** Create a series of sketches or screenshots to outline the user journey and interaction flow within your web application. Storyboards help visualize how users will navigate through your site.

iii. Design Templates:

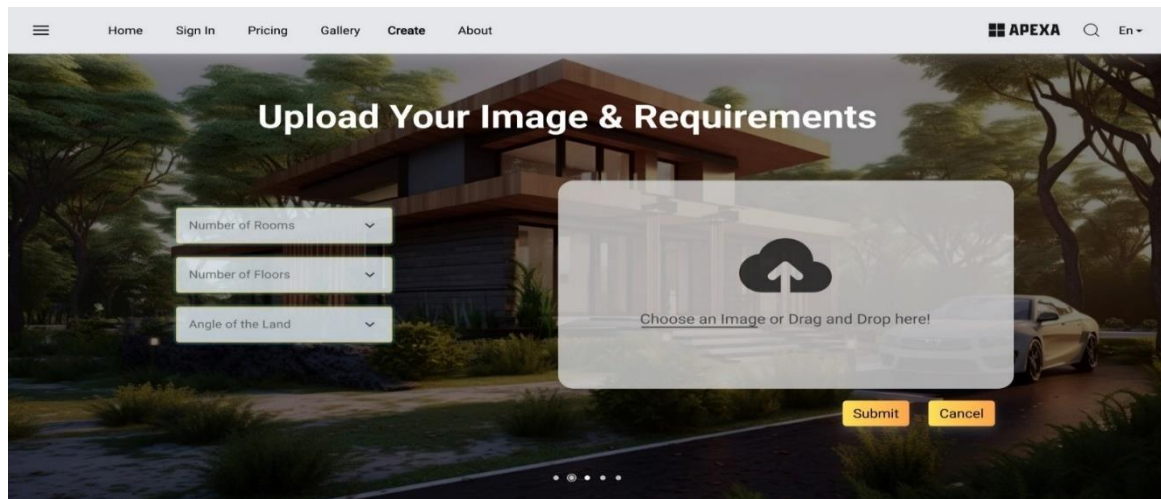
- Home Page



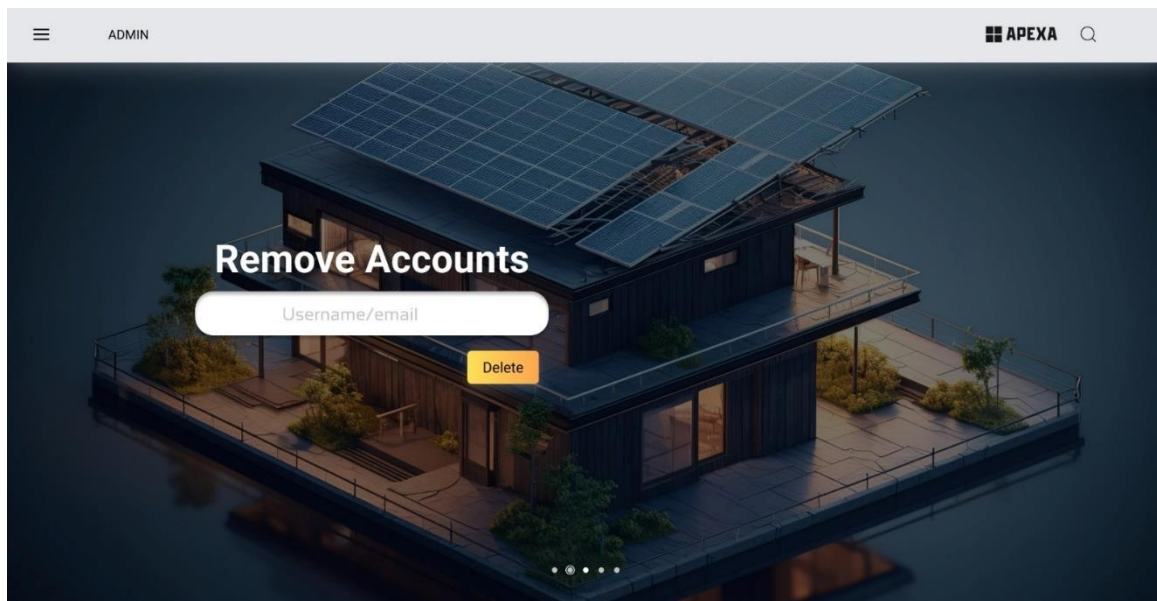
- Sign-in/Sign-up Page



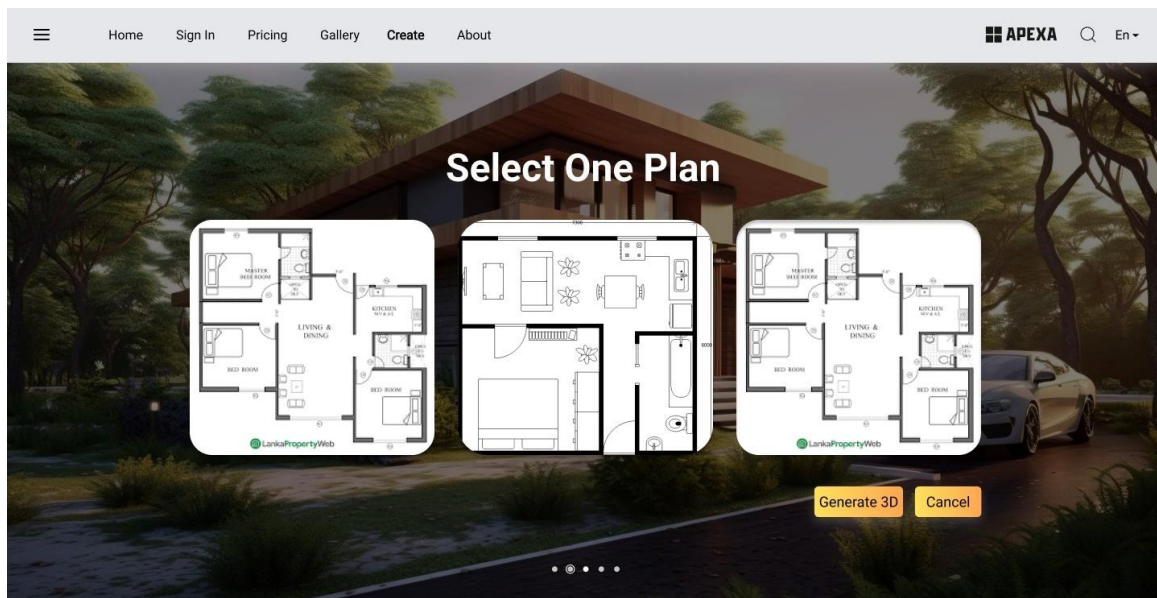
- Upload Image page



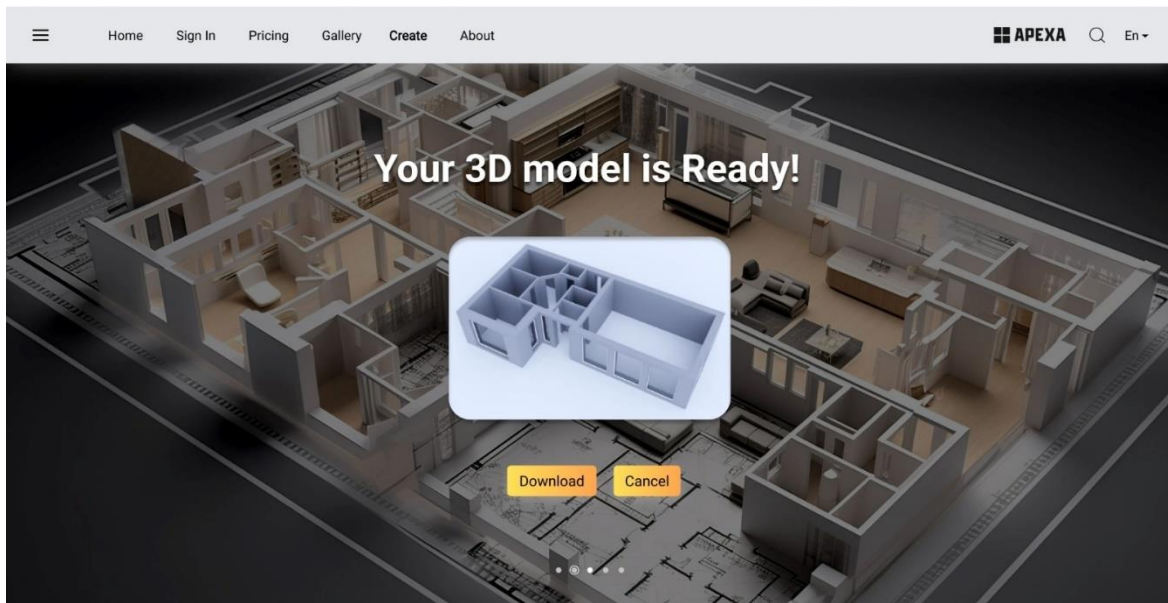
- Admin Page



- Generate Floor Plan Page



- 3D view Page



iv. Input Design aspects

Aspect	Description
Input Fields	Boxes where users type information.
Labels	Clear instructions next to input fields.
Validation	Checking input for correctness and providing feedback on errors.
Error Handling	Displaying helpful messages when users make mistakes.
Feedback	Providing immediate feedback on user input.
Accessibility	Ensuring all users, including those with disabilities, can use input fields easily.
Input Masks	Formatting input as users type (e.g., adding dashes to phone numbers).
Auto-Suggestions	Offering helpful suggestions as users type.
Mobile Optimization	Optimizing input fields for easy use on mobile devices.
Consistency	Maintaining uniformity in design and functionality across the project.

v. Output Design aspects

Aspect	Description
Visual Elements	Pictures, words, and graphics used to show information.
Layout	How everything is arranged on the screen.
Typography	The style and size of the text used in the design.
Color Palette	The colors chosen to make the design look good and easy to understand.
Icons and Symbols	Small pictures or signs that represent different things.
Consistency	Making sure everything looks the same across the whole design.
Responsiveness	Making sure the design works well on different devices, like phones and computers.
Accessibility	Making sure everyone, including people with disabilities, can use the design easily.
Interactive Elements	Buttons or links that people can click on or interact with.
Feedback Mechanisms	Giving people clear signals when they do something or make a mistake.






vi. Dialogue design aspects

Aspect	Description
Language and Tone	The words and how they sound, matching who we're talking to and how we want to sound.
Clarity	Making sure the words are easy to get, avoiding hard words or too much talk.
Brevity	Keeping it short and sweet, saying just what's needed without extra stuff.
Staying Relevant	Talking about things that make sense right now, giving info that matters.
Helping Users	Giving clear instructions to users, showing them how to do things.
Fixing Mistakes	Helping users when things go wrong, giving them a way to fix stuff.
Saying "Yes" and "No"	Saying when something's okay and when it's not, letting users know what's happening.
Keeping Things Alike	Making sure all the talking feels the same, so users know what to expect.
Open to Everyone	Making sure everyone can understand and use what we say, no matter who they are.

Section 05

Students Declaration

All the information stated in section 1,2, 3, and 4 are completed and ready to demonstrate in the examination of EC05.

Index Number	Registration Number	Name	Signature
5066	ICT/19/20/132	L. G. R. J. Lindapitiya	
5071	ICT/19/20/138	R. M.V. P. B. Udagama	
4957	ICT/19/20/016	B. M. C. B. K. Bandaranayake	
4997	ICT/19/20/059	A. G. N. D. Kaluwelgoda	
4944	ICT/19/20/002	T. M. M. M. B. Abeysinghe	

Date: 16/11/2024


Section 06

Supervisor Declaration

I agree / disagree with the information stipulated in this application form.

Name: ...Mr. K.H.A.Hettige.....

Department/ Organization: ...Department of Computing.....

Signature:

Date:16/11/2024.....