CODVYN

Build Stack Series / 2026 Edition

Integrated Build Stack Guide

Antigravity + Claude Code + Stitch

Google Antigravity **Claude Code** **Google Stitch**

Design in Stitch. Build in Antigravity. Precision-edit with Claude Code.
One IDE. Zero context switching. Full-cycle execution.

codvyn.in | instagram.com/codvyn

Google Antigravity IDE	**Anthropic Claude Code**	**Google Stitch**

This guide, published by Codvyn, documents how to build products faster using three tools in a single environment. Google Antigravity handles IDE orchestration and multi-agent task management. Claude Code handles precise, terminal-native code generation with full diff review. Google Stitch converts natural language prompts and wireframe images into production-ready HTML/CSS and React code in seconds. Together, they remove the gap between design, planning, and implementation.

Contents

codvyn.in | instagram.com/codvyn

01 Why This Stack Exists

The standard product development loop involves too many tools. A UI is designed in Figma, architecture is outlined in Notion or a separate doc, code is written in an editor, and deployment happens in a terminal somewhere else entirely. Every tool switch is a context switch, and every context switch costs time and introduces the possibility of drift between what was designed and what was built.

This stack is built to close that loop. Stitch handles design-to-code in seconds. Antigravity handles orchestration, planning, and multi-agent execution. Claude Code handles targeted, reviewable code generation with precise diff control. All three tools run inside or alongside a single IDE window, and they communicate through standard protocols: MCP, VS Code extension APIs, and the Claude Code CLI.

The result is a pipeline that compresses what normally takes days of tool hopping into a single, continuous session. This is not a theoretical workflow. The full cycle from text prompt to deployed application has been completed in under 30 minutes using exactly this stack.

What you will be able to do after reading this guide

- Generate production-ready UI screens from natural language in Stitch
- Connect Stitch to Antigravity using the Stitch API and MCP protocol
- Pipe Stitch-generated HTML and React code directly into Claude Code
- Use Antigravity's multi-agent system to parallelize architecture and implementation
- Maintain a clean Git history where design and code changes are versioned together

02 Tool Overview

Before going into individual tool depths, here is a precise summary of what each tool does inside this stack and how they relate to one another.

Tool	Role	Model	Access
Google Stitch	AI-powered UI design tool	Gemini 2.5 Flash / 2.5 Pro / Gemini 3	stitch.withgoogle.com
Google Antigravity	Agent-first IDE (VS Code fork)	Gemini 3 Pro + Claude Sonnet 4.5	Installed locally
Claude Code	Terminal-native agentic coder	Claude Sonnet 4.5 / Opus 4.5	npm install -g @anthropic-ai/claude-code

These tools are complementary. Stitch creates the visual layer. Antigravity orchestrates the agent layer. Claude Code handles the implementation layer. They are connected through the Model Context Protocol, allowing Claude Code to read and write Stitch designs as part of an automated workflow.

03 Google Stitch in Depth

Google Stitch is a web-based AI design tool launched at Google I/O in May 2025 as a Google Labs experiment. It was born from the acquisition of Galileo AI and rebuilt on Google's Gemini model infrastructure. It is available free at stitch.withgoogle.com and requires only a Google account.

How Stitch works

You provide Stitch with either a natural language prompt describing the interface you want, an uploaded image such as a whiteboard sketch, a screenshot of an existing UI, or a rough wireframe. Stitch processes the input through Gemini's multimodal pipeline and generates complete UI layouts with corresponding frontend code. The entire cycle takes between 10 and 45 seconds depending on complexity and the AI mode selected.

AI Modes

Mode	Model	Generation Limit	Strengths
Standard Mode	Gemini 2.5 Flash	Up to 350 generations per month	Fast layout generation, Figma export, theme controls
Experimental Mode	Gemini 2.5 Pro	Up to 50 generations per month	Higher fidelity, more detailed layouts, better prompt adherence

Core features

Stitch generates multi-screen designs, not just single views. You can design a full app flow and navigate between screens in the canvas. The Annotate feature, added in October 2025, lets you place comments directly on UI screens. Once submitted, Gemini reads the annotated screenshot and applies context-aware changes. The Theme sidebar lets you toggle light and dark mode, set primary colors, adjust corner radii, and change typography globally across all screens.

The Prototypes feature, introduced in December 2025 alongside Gemini 3 integration, allows you to link screens together and build interactive user flows. Click and input modes let you simulate real app behaviour before a single line of implementation code is written.

Export options

Export Method	Details
Copy to Figma	Standard Mode only. Pastes the entire layout with editable layers and auto-layout intact into any open Figma file.
View Code	Opens HTML + Tailwind CSS code for any screen. The code is clean, structured, and deployment-ready.
Export Archive	Downloads a zip containing all screen code and image assets.
Firebase Studio	Direct export to Firebase Studio via the Share button for cloud-connected deployment.
MCP / API	Programmatic access to projects, screens, and generated HTML via the Stitch MCP server.

Known limitations to plan around

Image upload for sketch-to-UI is inconsistent: Stitch sometimes asks for a text description instead of interpreting the image.

Best suited for single-screen generation. Multi-screen flows can break if you ask for changes across more than two screens at once.

The generated code uses HTML and Tailwind CSS. There is no direct React or SwiftUI export from the UI, though the MCP integration handles React conversion.

Figma export is Standard Mode only. Experimental Mode does not support Copy to Figma.

Generation limits reset monthly. Monitor usage if running Stitch in automated agent loops.

04 Google Antigravity in Depth

Google Antigravity is an agent-first IDE launched in November 2025. It is a fork of the open-source VS Code codebase, built around the concept of autonomous agent management rather than traditional prompt-driven chat assistance. It is powered by Gemini 3 Pro by default but supports multiple models including Claude Sonnet 4.5 and Claude Opus 4.5.

Mission Control

The core differentiator in Antigravity is Mission Control, its task management interface for AI agents. Rather than a single chat pane, Mission Control is a dashboard where you spawn, monitor, pause, and coordinate multiple agents running in parallel. Each agent has its own workspace context, terminal access, file system access, and browsing capability. You act as the architect, defining objectives and reviewing outputs.

Extension ecosystem

Because Antigravity is a VS Code fork, most existing VS Code extensions transfer directly. However, Antigravity uses the Open VSX extension registry rather than the Microsoft-proprietary VS Code Marketplace. Most popular extensions are available on Open VSX. For any extension not found there, download the VSIX file manually from the developer's site and install via the command palette.

Antigravity-specific capabilities

- Built-in browser for automated visual verification of running applications
- Multi-model support: switch between Gemini 3 Pro, Claude Sonnet 4.5, and others per task
- Integrated terminal with full agent access
- Visual diff viewer for reviewing multi-file changes before committing
- Native Stitch integration via agent context and MCP proxy

05 Claude Code in Depth

Claude Code is Anthropic's terminal-native agentic coding tool. It runs as a CLI, understands your entire codebase through context-aware file reading, and executes multi-file changes through natural language instructions. The VS Code extension integrates the CLI directly into Antigravity, creating a dedicated panel where code changes appear as inline diffs using the native diff viewer.

What makes Claude Code different from chat-based AI

Most AI coding assistants suggest code. Claude Code acts on code. It reads your project files autonomously, writes changes across multiple files, runs terminal commands, reads the output, and iterates based on what it finds. Every change is presented as a reviewable diff before it is written to disk. You stay in control of what lands, but you do not have to do the work.

Key capabilities

Capability	Description
Multi-file editing	Claude reads and modifies multiple files in a single task, maintaining consistency across the codebase.
Terminal integration	Claude can run commands, read output, and react to errors without you copying and pasting.
Context referencing	Use @filename or @terminal:name to reference files and terminal output directly in prompts.
Sub-agents	Claude can spawn sub-agents for parallel task execution within a session.
Custom slash commands	Store .md files in .claude/commands/ and call them with /commandname to run saved instructions.
Named sessions	Use /rename to name sessions and --resume to continue them from any terminal.
MCP connections	Claude connects to external tools, APIs, and services via the Model Context Protocol.
Context rewind	Press Escape twice to edit a previous message and roll back the code to that point.

06 Installation and Prerequisites

Install the tools in the order below. Each step has dependencies on the previous one.

01 **Install Node.js (required for Claude Code)**

Open a terminal and run: node --version. You need Node.js version 18 or higher. If Node.js is not installed, download it from nodejs.org or use a version manager such as nvm. This is a hard requirement. The Claude Code CLI will not run without it.

02 **Install Claude Code CLI**

Run: npm install -g @anthropic-ai/claude-code. After installation, run claude in any terminal to authenticate with your Anthropic account. You need a Claude Pro, Max, Team, or Enterprise subscription. Once authenticated, you can run claude from the Antigravity integrated terminal and the IDE extension will detect and install itself automatically.

03 **Install Google Antigravity IDE**

Download Antigravity from the official site. It is available for macOS, Windows, and Linux. Sign in with your Google account. Antigravity is free during public preview for users on Google AI Pro or Ultra plans. After installation, open a project folder and run claude in the integrated terminal to activate the Claude Code extension panel.

04 **Set up Stitch access**

Go to stitch.withgoogle.com and sign in with your Google account. No installation is required. Stitch is entirely web-based. You need a Google account and an active internet connection. Familiarise yourself with the interface by generating a test UI before setting up the API connection.

05 **Create and configure your project workspace**

Create a new folder for your project. Open it in Antigravity. Initialise a Git repository with git init. Create a CLAUDE.md file at the root. Create a .claude/ directory for configuration and custom commands. This is the folder structure all tools will reference.

Stitch is free during Google Labs beta

As of February 2026, Stitch is free with monthly generation limits: 350 in Standard Mode and 50 in Experimental Mode.

The Stitch MCP server is also free of charge.

Limits and pricing are subject to change as Stitch moves out of Google Labs.

07 Connecting Stitch to Antigravity via API

Antigravity can connect to Stitch directly through its agent context system. When an Antigravity agent is given access to the Stitch MCP server, it can list your Stitch projects, retrieve screens, download HTML assets, and generate new screens from text prompts, all from within the IDE without opening a browser.

Method 1: Using gcloud Application Default Credentials (ADC)

This is the recommended method for teams and for production use. It authenticates through Google Cloud rather than a personal API key.

```
# Step 1: Install gcloud CLI from cloud.google.com/sdk
gcloud auth login

# Step 2: Set your Google Cloud project
export PROJECT_ID="your-gcp-project-id"
gcloud config set project $PROJECT_ID
gcloud auth application-default set-quota-project $PROJECT_ID

# Step 3: Enable the Stitch MCP API on your project
gcloud beta services mcp enable stitch.googleapis.com --project=$PROJECT_ID

# Step 4: Grant the required IAM role to your account
gcloud projects add-iam-policy-binding $PROJECT_ID \
--member="user:your-email@gmail.com" \
--role="roles/serviceusage.serviceUsageConsumer"

# Step 5: Authenticate ADC
gcloud auth application-default login
```

Once authenticated, the Antigravity agent can communicate with the Stitch MCP server using your ADC credentials automatically. No key management is required after setup.

Method 2: Using a Stitch API Key (personal use)

The API key method is simpler to set up and is appropriate for personal projects. Note that the API key option may not be available to all accounts during the current beta period. If the API Keys section is not visible in your Stitch settings, use ADC instead.

1	Go to stitch.withgoogle.com and sign in
2	Click your profile icon in the top-right corner
3	Select Stitch Settings from the dropdown
4	Navigate to the API Keys section
5	Click Create Key and copy the generated key
6	Store it securely, you will not be able to see it again

Once you have the key, configure it in the Antigravity agent environment:

```
# Set the API key as an environment variable
export STITCH_API_KEY="your-api-key-here"
```

```
# Or add it permanently to your shell profile
echo 'export STITCH_API_KEY="your-api-key-here"' >> ~/.zshrc
source ~/.zshrc
```

Installing the Stitch Gemini CLI extension

The Gemini CLI extension enables natural language interaction with Stitch directly from any terminal, including the one inside Antigravity. Install it with:

```
gemini extensions install https://github.com/gemini-cli-extensions/stitch --auto-update
```

After installation, configure the extension with your project ID:

```
# Replace the placeholder in the extension config with your actual project ID
sed -i.bak "s/YOUR_PROJECT_ID/$PROJECT_ID/g" \
~/.gemini/extensions/Stitch/gemini-extension.json
```

Once set up, you can interact with Stitch from inside Antigravity's terminal:

```
# List your Stitch projects
/stitch What Stitch projects do I have?

# Get all screens from a specific project
/stitch Give me all the screens of project 3677573127824787033

# Generate a new screen from a prompt
/stitch Design a mobile onboarding flow for a fintech app with dark theme, using Gemini 3 Pro

# Download the HTML for a specific screen
/stitch Download the HTML of screen 6393b8177be0490f89eb8f2c1e4cfb37
```

API key availability note

As of early 2026, some Stitch accounts show the API Keys section only after enabling the Stitch MCP API on a Google Cloud project.

If you do not see API Keys in Stitch Settings, complete the ADC setup first. The option will appear after gcloud permissions are configured.

Never commit your STITCH_API_KEY to version control. Add it to your .gitignore or use an environment variable manager.

08 Connecting Stitch to Claude Code via MCP

Claude Code connects to Stitch through the stitch-mcp package, a CLI tool that bridges your IDE's coding agent to the Stitch platform. Once connected, Claude Code can read your Stitch project screens, serve them locally for visual preview, and convert the generated HTML and Tailwind CSS into React components.

Install stitch-mcp

```
# Initialise the stitch-mcp client and authenticate
npx @_davideast/stitch-mcp init

# This command sets up:
# - Authentication with your Stitch account
# - MCP client configuration
# - Project token management with automatic refresh
```

Serve Stitch screens locally

Once connected, you can run a local Vite development server that serves all screens from a Stitch project. This lets Claude Code inspect the running UI in a browser and verify visual output without leaving the IDE.

```
# Serve all screens from a project on a local dev server
npx @_davideast/stitch-mcp serve -p

# The server starts on localhost with hot reload
# Claude Code can then navigate to it via the Antigravity browser agent
```

Generate a site structure from Stitch screens

```
# Build an Astro site by mapping Stitch screens to routes
npx @_davideast/stitch-mcp site -p

# This generates a navigable multi-page site
# Each screen becomes a route, preserving layout and theming
```

Using the MCP proxy with Claude Code

The proxy command bridges Claude Code's agent to the Stitch MCP tools with automatic token refresh. This is what enables Claude Code to read Stitch design data programmatically as part of an agent workflow.

```
# Start the MCP proxy (run this before starting Claude Code session)
npx @_davideast/stitch-mcp proxy

# Then in Claude Code, reference Stitch designs directly:
# 'Read the Stitch screen for the dashboard view and convert it
# to a React component using our existing Button and Card components'
```

Working pattern: Stitch to Claude Code

1. Generate the UI screen in Stitch using a natural language prompt
2. Run npx @_davideast/stitch-mcp serve to preview screens locally
3. Start the MCP proxy in a separate terminal
4. In Claude Code, reference the Stitch screen and ask for React component conversion
5. Claude reads the HTML, maps it to your design system, and writes the component
6. Review the diff, approve it, and move to the next screen

09 Configuration Reference

Key files and folders

File or Folder	Purpose
CLAUDE.md	Project-level context file. Claude Code reads this at every session start. Put architecture decisions, stack choices, and naming conventions here.
.claude/settings.json	Claude Code permissions, model selection, and tool access controls.
.claude/commands/	Custom slash commands as .md files. Call them with /commandname inside any Claude Code session.
.claude/rules/	Persistent rules that Claude follows across all sessions in this project. Use this for code style enforcement.
.env	Environment variables including STITCH_API_KEY. Never commit this file.
.gitignore	Must include .env and any files containing API keys or secrets.
stitch-mcp.config.json	stitch-mcp project configuration. Created by npx @_davideast/stitch-mcp init.

Antigravity agent mode selection

Mode	When to Use
Plan Mode	Produces an architecture breakdown and awaits your approval before executing. Use this at the start of a new feature.
Agent Mode	Full autonomous multi-step execution. Use for complex tasks once the architecture is agreed.
Edit Mode	Quick targeted file edits. Use for small changes where agent overhead is unnecessary.

Claude Code permission settings

By default, Claude Code requests permission before terminal commands, file creation, and edits. For trusted projects, you can configure auto-accept for low-risk operations in .claude/settings.json. Always keep manual review enabled for files touching authentication logic, environment variables, database schemas, and deployment configuration.

10 Full Workflow Execution Model

The following is the complete production workflow for this stack. All phases occur inside or alongside the Antigravity workspace.

01

Architecture planning in Antigravity Plan Mode

Before writing any code, use Antigravity's Mission Control in Plan Mode. Describe your product: the application type, target platform, performance constraints, and technology preferences. The agent generates a proposed folder structure, technology stack, and dependency list. Review and confirm it. Save the output to CLAUDE.md so every subsequent AI session inherits the context automatically.

02

UI generation in Stitch

Open stitch.withgoogle.com. Use Standard Mode for fast iteration or Experimental Mode for higher fidelity. Write structured prompts describing your screens: platform (mobile/web), layout structure, color palette, key components, and interaction style. Generate all primary screens of your product. Use the Annotate feature to refine specific elements. Use the Theme sidebar to align the output with your brand. Save the project and note your project ID.

03

Bridge Stitch into the IDE

In the Antigravity terminal, run npx @_davideast/stitch-mcp init to connect the IDE to your Stitch project. Then run npx @_davideast/stitch-mcp serve to start a local dev server with all your screens. Antigravity's browser agent can now navigate your generated UI and verify visual output. Start the MCP proxy in a separate terminal tab so Claude Code has programmatic access to the design data.

04

Component generation with Claude Code

Open the Claude Code panel in Antigravity via the spark icon. Reference a specific Stitch screen and ask Claude to convert it to a React component using your project's design system. Specify the framework, version, and target file explicitly. Claude reads the HTML from Stitch via the MCP proxy, maps it to your component library, and presents a diff. Review and accept each component before moving to the next screen.

05

Parallel agent work in Mission Control

While Claude Code handles component implementation, use Antigravity Mission Control to run parallel agents on other tasks: test generation, API integration, database schema setup, or documentation writing. Assign each agent a clearly bounded area of the codebase to avoid file conflicts. Monitor progress in the Mission Control dashboard.

06

Iteration and visual verification

As components are built, the Antigravity browser agent navigates the running application and compares the rendered output against the Stitch reference screens. Discrepancies are reported back to Claude Code as context, and corrections are applied through the standard diff review process. Commit each verified component before moving to the next.

07

Deployment

Use Claude Code to write commit messages based on actual diff content, create pull requests with descriptions that explain the design-to-code journey, and generate deployment configuration files. For Google-ecosystem projects, the Stitch Share button provides direct export to Firebase Studio for cloud deployment without leaving the pipeline.

11 Prompt Engineering and Tips

The quality of output from all three tools in this stack depends directly on how well you write prompts. The following tips are specific to this stack and have been tested against real workflows.

Prompting Stitch effectively

Practice	Explanation
Specify the platform	Always state whether you want a mobile or web UI. Stitch produces very different layouts depending on this. Example: 'iOS-style mobile app' vs 'desktop web dashboard'.
Describe the layout hierarchy	Name the layout structure explicitly. Example: 'sticky header, two-column content area, sidebar on the right with filter controls, card grid in the main area'.
Include visual direction	State the aesthetic: clean and minimal, dark mode, glassmorphism, flat design. Stitch responds well to design vocabulary. Avoid vague terms like 'modern' alone.
List key components	Name the UI components explicitly: 'nav bar, search bar, tab switcher, transaction list items with icon, date, amount, and status badge'.
Specify constraints	State what you do not want: 'no heavy gradients', 'accessible text sizes', 'one-hand usage optimised'. Stitch respects explicit constraints.
Use follow-up prompts	The first generation is a starting point. Refine it: 'Move the app name to the left side of the header', 'Change the background to dark gray', 'Add WCAG 2.1 contrast compliance'.
Break complex flows apart	Do not ask for a full 10-screen app in one prompt. Generate each screen separately and refine before moving on. Stitch handles 1 to 3 screens well per generation.

Prompting Claude Code effectively

Practice	Explanation
Reference exact file names	Say 'edit src/components/Header.tsx' not 'edit the header'. Ambiguous references cause Claude to guess.
Specify framework and version	Always include the full framework context: 'React 18, TypeScript 5, Tailwind CSS 3.4'. AI behaviour differs significantly across versions.
Limit scope per request	One clear task per prompt produces tighter, easier-to-review diffs. Do not bundle unrelated changes.
Use CLAUDE.md for context	Put architecture decisions, naming conventions, and tech stack in CLAUDE.md so you do not repeat them each session.
Reference Stitch screens	When asking Claude to implement a component, say: 'Using the Stitch dashboard screen as the visual reference, implement the card component in src/components/DashboardCard.tsx'.

Validate before executing	For any prompt touching auth, database schemas, or env config, read the full diff carefully before accepting.
Use Escape twice to rewind	If Claude generates something wrong, press Escape twice to edit a previous message and roll the code back to that state before trying again.

Example: a well-structured Stitch prompt

Platform: iOS mobile app, one-hand usage optimised

Screen: Habit tracker home screen

Layout: sticky header with app name left-aligned and streak count right-aligned, daily habit list with checkbox and streak indicator per item, floating action button bottom-right

Visual: dark background (#1A1A2E), purple accent (#7B61FF), clean minimal, no gradients

Constraints: WCAG 2.1 AA contrast, accessible text sizes, no heavy shadows

Example: a well-structured Claude Code prompt

In src/components/HabitCard.tsx, implement the habit card component.

Use the Stitch home screen as the visual reference (available via MCP proxy).

React 18 with TypeScript, Tailwind CSS. Use the existing Checkbox component from src/ui/.

Props: name: string, streak: number, isCompleted: boolean, onToggle: () => void.

Do not modify the existing theme tokens in tailwind.config.ts.

12 Multi-Agent Workflows

Antigravity's Mission Control is designed for parallel agent execution. Used well, it turns a linear development process into a concurrent one, with different agents handling different concerns simultaneously.

How to structure parallel work

Agent Role	Responsibility
Agent 1: Architecture	Plan Mode. Generates folder structure, component hierarchy, and data model. Outputs to CLAUDE.md.
Agent 2: Component build	Claude Code. Converts Stitch screens to React components one at a time, with diff review at each step.
Agent 3: Test generation	Runs in parallel to Agent 2. Writes unit and integration tests for each completed component.
Agent 4: API integration	Handles backend wiring, API calls, and state management while the UI components are being built.

Conflict avoidance

The most common issue with parallel agents is file conflicts. Assign each agent a clearly bounded area of the codebase. Component agents should only touch src/components/. API agents should only touch src/api/ and src/hooks/. Test agents should only write to the tests/ directory. If two agents need to modify the same file, run them sequentially rather than in parallel for that specific task.

Switching models per task

Antigravity lets you assign different models to different agents. Use Gemini 3 Pro for broad planning and architectural tasks where large context is important. Use Claude Sonnet 4.5 in Claude Code for precise, reviewable code generation. Use Gemini 2.5 Flash for fast, repetitive tasks like renaming, reformatting, or generating boilerplate.

Do not run agents with overlapping file targets simultaneously

Antigravity provides visual conflict resolution, but it is slower than avoiding conflicts in the first place.

Use branch isolation: each agent works on its own Git branch, and you merge when the outputs are clean.

For critical files like auth logic, database schemas, and environment config, never assign them to agent tasks. Handle them manually.

13 Version Control and Git Practices

Working with AI-generated code at speed makes Git hygiene more important, not less. The following practices protect you from difficult-to-reverse changes and make your project history readable by any developer who joins later.

Practice	Why It Matters
Commit after each accepted diff	Each approved Claude Code change = one commit. Batching many changes makes rollback harder and diffs unreadable.
Write AI-assisted commit messages	Tell Claude Code: 'Write a commit message for these changes.' It reads the actual diff and writes an accurate description.
Branch per feature	Each feature or screen implementation lives on its own branch. Merge to main only after manual diff review.
Never merge without review	AI-generated code is correct most of the time, not always. Review every branch before merging to main regardless of how confident the agent was.
Tag design checkpoints	When you finish all Stitch screens for a feature, create a Git tag. This gives you a clean reference point before implementation begins.
Store Stitch exports in Git	Save downloaded Stitch HTML exports to a /stitch-exports/ folder in your repo. This records the design intent alongside the implementation code.
Use .gitignore carefully	Always ignore .env, node_modules, .claude/cache, and any files containing API keys or credentials.

14 Troubleshooting

Issue	Resolution
Stitch API Keys section not visible	You need to enable the Stitch MCP API on a Google Cloud project first. Complete the gcloud setup steps in Section 07. The API Keys option appears after cloud permissions are in place.
npx @_davideast/stitch-mcp init fails	Ensure you are authenticated with gcloud: run gcloud auth application-default login and retry. If using an API key instead, check that the STITCH_API_KEY environment variable is set in the same shell session.
Claude Code extension not appearing in Antigravity	Run the claude command from within the Antigravity integrated terminal. The extension auto-installs when it detects the CLI inside the IDE. If it still does not appear, run Developer: Reload Window from the command palette.
MCP proxy connection drops mid-session	The proxy uses time-limited signed URLs for asset downloads. Restart the proxy with npx @_davideast/stitch-mcp proxy. Always fetch fresh download URLs before downloading assets rather than caching them.
Stitch image upload does not interpret sketch	This is a known Stitch limitation. If the upload asks for a text description instead of interpreting your image, describe the layout in the text prompt and use the image as a reference alongside it rather than the sole input.
Stitch generates only one screen instead of multiple	Break your app into individual screens and generate one at a time. Asking for a full multi-screen app in a single prompt frequently produces incomplete results. Use follow-up prompts to add screens.
High memory usage in Antigravity with multiple agents	Close agents you are not actively monitoring. Restart the IDE if performance degrades significantly. Long conversation histories in Claude Code also increase memory. Start fresh sessions for new features and load context from CLAUDE.md.
Figma export button not visible in Stitch	Copy to Figma is only available in Standard Mode. Switch from Experimental Mode if you need Figma export. In Experimental Mode, use the Export Archive option instead.
Tailwind tokens missing after React conversion	The Stitch HTML export uses a localised Tailwind config. When Claude Code converts it to React, instruct it to merge the localised Tailwind tokens with your project's tailwind.config.ts rather than hardcoding values.

15 Quick Reference: Commands and Shortcuts

Stitch MCP commands (inside Antigravity terminal)

Command	Action
/stitch What Stitch projects do I have?	List all projects in your Stitch account
/stitch Give me all screens of project	List all screens in a project
/stitch Download the HTML of screen	Download generated HTML for a screen
/stitch Download the image of screen	Download a screenshot of a screen
/stitch Design a using Gemini 3 Pro	Generate a new screen from a text prompt
/stitch Enhance this prompt:	Ask Stitch to expand and improve a prompt
/mcp list	List all available MCP tools in the session

stitch-mcp CLI commands

Command	Description
npx @_davideast/stitch-mcp init	Authenticate and set up MCP client config
npx @_davideast/stitch-mcp serve -p	Start local Vite server with all project screens
npx @_davideast/stitch-mcp site -p	Build an Astro site from screen-to-route mappings
npx @_davideast/stitch-mcp proxy	Start MCP proxy for Claude Code agent access

Claude Code slash commands

Command	Description
/ide	Connect Claude Code CLI to the current IDE
/mcp	Open MCP server authentication and management
/terminal-setup	Configure terminal keybindings for multi-line input
/rename	Name the current session for later resumption
/resume	Resume a named session
/model	Switch the active model mid-session
/stats	View usage statistics and model history

Key Antigravity shortcuts

Shortcut	Action
Ctrl/Cmd + Shift + X	Open Extensions panel
Ctrl/Cmd + Shift + P	Open Command Palette
Ctrl/Cmd + `	Open integrated terminal
Escape + Escape	Rewind to previous message in Claude Code (context rollback)
Ctrl/Cmd + N	Start a new Claude Code conversation

gcloud commands for Stitch API setup

Command	Purpose
gcloud auth login	Authenticate your Google account with gcloud
gcloud auth application-default login	Set up Application Default Credentials
gcloud config set project	Set the active Google Cloud project
gcloud beta services mcp enable stitch.googleapis.com	Enable the Stitch MCP API
gcloud projects add-iam-policy-binding ...	Grant serviceusage.serviceUsageConsumer role

A note from Codvyn

This guide is a living document. Stitch, Antigravity, and Claude Code are all evolving fast.
Follow codvyn.in and instagram.com/codvyn for updated workflows, tutorials, and new stack guides.
If you find anything outdated or have a workflow improvement to share, reach out through the Codvyn site.