SpecForge Buildathon

Participant Handbook

Cynergy Coding Club, RUAS Peenya | 11:15 to 17:00 | Solo | Free | Build with AI

1. Overview

Item	Detail
Event	SpecForge Buildathon: a one-day, guided "build with AI" event
Time and place	Thursday, 29 October 2026. Check-in at 11:00, event runs 11:15 to 17:00, RUAS Peenya. The hall will be announced by the organisers.
Format	Solo. Everyone builds and submits their own app. Sitting together is fine; sharing code is not.
Who can join	Students of any branch and any year. No prior coding experience is needed on the guided track.
Cost	Free.
AI tools	Any AI tool, model or assistant is allowed. Say which ones you used in your README.
How you are scored	Out of 100: your output file (80) and your spec document (20), scored automatically on a live leaderboard. The final ranking comes from the organisers running your code.
Certificates	E-certificates are confirmed.

Background. Every course has a course specification: a formal document listing the summary, credits, Course Outcomes (COs), unit-wise syllabus, CO-PO-PSO mapping, teaching hours, assessment plan, skills and resources. Accreditation bodies (NBA, under the Washington Accord) check these closely. In practice, specs are edited by hand, copied between versions and drift out of sync with the question-paper blueprint. CO-PO mappings often disagree with the cognitive level of the CO, COs appear in one table but not another, and summaries no longer match the syllabus. Your app must fix this. The problem comes from our Head of Department, and the best ideas may be reused by the department.

2. The problem

Build a web application in which an Author uploads a question-paper blueprint and an existing course specification, uses AI to generate a corrected course specification in the official HTML format, edits it, and sends it through Reviewer and HOD approval to a final, downloadable PDF. Your app must also write its result to a fixed JSON file (Section 6) so that it can be scored.

The generated specification must:

1. Follow the official HTML template exactly. Use the Integrated (theory + lab) template for courses with a practical component and the Non-Lab template otherwise.
2. Take Course Outcomes and the unit-wise syllabus from the blueprint.
3. Take every other section from the existing course specification.
4. Carry a revised CO-PO-PSO mapping that is consistent with each CO’s Bloom’s level and the Programme Outcomes in Annexure A. The mapping in the existing spec is known to be wrong in several places. Do not copy it blindly.
5. Fix internal inconsistencies: COs missing from or extra in a table, a summary that does not match the syllabus, hours or weightages that do not add up, a wrong template choice, leftover template sample text, typos in references.

Input kit

#	File	What it is	Role
1	1. SpecEditor-IntegratedLabCourses.html	Official editor/template for theory + lab courses (pre-filled with a sample course)	Output format
2	2.SpecEditor-Non_Lab_Courses.html	Official editor/template for theory-only courses (same sample)	Output format
3	3.Sample_Blueprint.doc	Question-paper blueprint for CSD103B	Source of truth for COs and units
4	4.Sample Course Spec.docx	Existing course specification for CSD103B	Source for all other sections
5	Annexure-A_PO_PSO_and_Blooms.docx	PO and PSO statements and the Bloom's verb list	Reference for mapping

The sample text in the two HTML templates (Computer Vision, AIC304A) is placeholder content. None of it may survive into your output. Do not hard-code the sample course: your app must work on a course it has never seen.

3. Source-of-truth rules

The template decides how the spec looks. The blueprint and the existing spec decide what it says.

Output section	Blueprint present	No blueprint
Header (title, code, type, department, faculty)	Existing spec (code and title must match the blueprint)	Existing spec
1. Course Summary	Existing spec, revised so it describes the blueprint's units	Existing spec
2. Course Size and Credits	Existing spec	Existing spec
3. Course Outcomes	Blueprint	Existing spec
4. Course Contents (unit text)	Blueprint	Existing spec
4. Lab exercises per unit (integrated only)	Existing spec	Existing spec
5. CO-PO-PSO Map	Re-derived by your app (AI + rules), never copied as is	Re-derived
6. Teaching and Learning Methods	Existing spec, totals recomputed	Existing spec, totals recomputed
7. Assessment (CO focus table)	Existing spec, corrected so every CO is assessed and no non-existent CO appears	Same
8. Achieving COs	Existing spec (lab courses must list the laboratory for practical skills)	Existing spec
9. Course Resources	Existing spec, typos and broken references cleaned	Existing spec

Template selection. A non-zero Practical value in the credit structure (for example 3:0:1), or a course type containing "laboratory", means the Integrated template. Otherwise use the Non-Lab template. The Author may override, with a reason.

Template structure wins. Keep section numbering, headings, table layout, CSS and the "Download as PDF" behaviour. Content in the existing spec that has no place in the template (for example a "10. Course Organization" section) is not added to the body. Report it in the validation report instead.

Conflicts are reported, not hidden. When the two sources disagree, or the existing spec contradicts itself, apply the rule above and record the conflict (what, where, which value you chose, why) in a validation report.

4. What your app must do

Role	Can do	Cannot do
Author	Upload inputs, run AI generation, edit any section, see the validation report, submit to Reviewer, respond to comments	Approve
Reviewer	View spec and validation report, comment per section, return to Author or forward to HOD	Edit content
HOD	View everything, approve, or return to Author with remarks	Edit content

Workflow states: Draft, Submitted, Under Review, Pending HOD, Approved, plus Returned (after which the Author revises and resubmits). Every state change is logged with user, role, time and comment. Until a spec is Approved, the PDF carries a DRAFT watermark.

Core features

- Roles. A role switcher (Author, Reviewer, HOD) is enough for today. Permissions must be checked on the server, not only hidden in the screen.
- Upload and parse the blueprint and the existing spec (.doc, .docx, .pdf, .html) into structured data.
- Template selection by the rule in Section 3, with a manual override.
- AI generation: merge the sources, re-derive the CO-PO-PSO map with a one-line justification for every non-empty cell, and rewrite the summary to match the units.
- Rule-based validation (plain code, not only AI): CO counts agree across Sections 3, 5 and 7; every CO is mapped and assessed; no CO appears that is not defined; hours add up; weightages total 100%; no template sample text remains; PSO columns match the programme.
- An editor, review comments and the approval workflow.
- Output: the spec as HTML and PDF (browser print-to-PDF is fine), the validation report, and the scoring file specforge_output.json.

CO-PO mapping rules

A value must be justified by the CO's verb (Bloom's level) and by evidence in the syllabus and assessment. If you cannot justify it in one sentence, it is wrong.

- Scale: 3 = very strong, 2 = strong, 1 = moderate, blank = none. No other values.
- Every CO maps to at least one PO. Every CO in Section 3 has exactly one row in Section 5, and there are no extra rows.
- A CO cannot map more strongly to a PO than its Bloom's level supports (for example, an L2 "Explain" outcome with PO-3 design = 3 is wrong).
- PO-5 needs real tool use (lab, programming). PO-6 to PO-10 need explicit evidence in the syllabus or assessment. No evidence means blank.

- A CO mapped strongly (2 or 3) to PO-3 or PO-5 must be assessed in a practical or assignment component in Section 7.
- Use exactly the PSO columns the template has (PSO-1, PSO-2).

5. Required tech stack

Your app is built so that the department can plug it into its own platform later. Build on this stack so that integration stays easy.

Layer	Use
Language	TypeScript
Runtime	Node.js 20 LTS or newer
Frontend	React 18 single-page app, built with Vite
Backend	Node.js REST API with Express
Validation	Zod schemas (also use Zod to check your own output file)
Optional libraries	React Router, TanStack Query, Axios, React Hook Form, Tailwind CSS, date-fns

API conventions (follow exactly)

- REST over HTTPS with JSON bodies, under the base path /api/v1.
- Success: { "success": true, "data": { ... } }
- Error: { "success": false, "error": { "message": "Readable text", "code": "MACHINE_CODE" } }
- Standard status codes: 400 invalid input, 401 not signed in, 403 not permitted, 404 not found, 409 conflict, 5xx server error.
- Dates are ISO 8601 strings in UTC. IDs are opaque strings. Files go up as multipart/form-data.
- Bring your own AI key (BYOK): your app has a Settings panel where you paste your own Gemini or OpenRouter key. The key stays in your browser, is never saved on the server, and never goes into source code, prompts, the output file or your repository.
- Every permission check happens on the server.

Not needed today: Microsoft sign-in (use the role switcher, with seeded users identified by email) and PostgreSQL/Prisma (in-memory or a JSON file is fine). Keep your own data separate from any other system.

6. The scoring file: specforge_output.json

Your app must produce one JSON file for each input it processes. The file must follow the schema in specforge_output.schema.json (included in this kit). A file that does not match the schema scores 0, and you will be shown the format errors so you can fix them. No extra keys are allowed.

Key	Content
schema_version	Always "1.0"
input_id	When you run on the sample kit, this must be exactly "SAMPLE". Any other value scores 0.
template	"integrated" or "non_lab"
course	title, code, department, credits_ltp (as "3:0:1"), course_type, summary
course_outcomes	List of { id: "CO-1", statement, bloom_level (1 to 6) }
units	List of { number, title, lab } where lab is text or null

Key	Content
co_po_pso_map	{ "CO-1": { "PO-1": { "level": 3, "justification": "..." }, ... } }. A missing cell means blank. Columns are PO-1 to PO-11 and PSO-1, PSO-2.
teaching_hours	Use exactly these keys: lectures, lab, tests_exams, total
assessment	components: list of { name, weight_percent }. co_coverage: { "CO-1": ["component name", ...] }
defects	List of { code, location: { source, section, refs }, description, fix }. See below.

Reporting defects

Each problem your app finds in the inputs is one entry in defects. Choose the code from this fixed list:

```
CO_MISSING_IN_TABLE  CO_EXTRA_IN_TABLE  CO_ASSESSMENT_MISMATCH  PSO_COLUMN_MISMATCH
MAPPING_BLOOM_MISMATCH  MAPPING_UNSUPPORTED  WRONG_TEMPLATE  PLACEHOLDER_TEXT
SUMMARY_SYLLABUS_MISMATCH  LABEL_OR_TITLE_MISMATCH  HOURS_INCONSISTENT
WEIGHTAGE_INCONSISTENT  TYPO_OR_FORMAT  SECTION_NOT_IN_TEMPLATE  MARKS_INCONSISTENT
OTHER
```

- location.source says where the problem lives: "spec" (the existing course spec), "blueprint", or "template" (for example the wrong template choice or leftover template sample text).
- location.section is the section number (1 to 12) of the spec where the problem is. Leave it null if it does not belong to one section.
- location.refs lists the items involved, such as "CO-5", "PSO-3" or "PO-10".
- Report real problems only. Padding the list with made-up defects costs points.

7. How you are scored (100 points)

Two files, graded by fixed rules (no AI): the output file specforge_output.json (80 points) and the spec document spec.html (20 points).

Output file part	Points	What is compared
CO-PO-PSO matrix	24	Every cell against the expert map. Off by one level earns half credit.
Defects found	20	The real problems in the inputs. Junk reports lose points.
Course details	8	Title, code, department, credits, and no leftover template text
Course outcomes	8	The set of COs, their wording and their Bloom’s level
Hours and assessment	8	Teaching hours, weightages total 100%, and CO coverage
Template choice	4	Integrated or Non-Lab
Units	4	Unit titles
Summary	4	The summary covers the syllabus themes

Spec document check	Points	What is checked
Template sections	4	The nine numbered sections of the official template, in order

Spec document check	Points	What is checked
No leftover sample text	3	Nothing left from the blank template's sample course
Right template	3	Integrated (with lab) or Non-Lab, matching the course
Course details	3	Course code, title and credits as L:T:P
Course outcomes	3	Every CO appears in the document
Map matches your JSON	4	The CO-PO-PSO table in the document agrees with your output file

You see your points for each part, never which exact cell or defect was wrong. That is deliberate: you have to understand the problem, not guess your way to a score. Partial credit is given almost everywhere. The XP board shown on screen is just for fun and never affects your score.

8. Submitting and the leaderboard

Everything happens on the participant portal: sign in with Google, then add your GitHub repository link, or plan to hand in a .zip of your project instead. The portal also has your getting-started steps, every level prompt with a copy button, and the whole kit.

How to submit

1. Start your app (npm run dev). In a second terminal run: npm run verify -- gemini YOUR_KEY (or openrouter / anthropic).
2. This calls your app exactly the way the organisers will, saves outputs/specforge_output.json and outputs/spec.html, and checks the output file.
3. In Submit, upload both files, or one .zip of your project (page 9), then press Submit for scoring.
4. Your points for each part appear straight away. Improve your app and upload again as often as you like.

Rules of the leaderboard

- You get a score out of 100 within seconds, with points for each part.
- Improve and resubmit as often as you like until 15:45. Only your best score counts. You can submit every 3 minutes.
- The leaderboard freezes at 15:45 sharp. Then the organisers run your code: your last commit before 15:45, or your latest zip if newer (npm install, npm start, POST /api/v1/generate with the sample files). That run is graded the same way and decides the final board.
- So keep npm run verify passing. A score your code cannot reproduce does not count.
- Your README must say which AI tools you used, which track you followed.

9. Schedule

Time	Block	What happens
11:00	Check-in	Sign the attendance sheet. Laptop, charger and Wi-Fi check. Mentors check your pre-work.
11:15	Briefing (15 min)	Problem, roles, rules, how scoring works, and the score of the organisers' reference build (the bar to beat)
11:30	Build-with-AI primer + L0	Pick your tool, connect GitHub, first push
12:00	L1 Reader	Live demo, then you build
12:25	L2 Template	Includes a 3-minute speed-run by the live coder

Time	Block	What happens
12:45	L3 Mapper	Runs into the working lunch. You also produce your first scoring file.
about 13:00 to 13:40	Working lunch	Mentors check on anyone who is stuck
13:40	L4 Inspector + Bug Hunt	Validation report, then a short bug hunt
14:05	L5 Workflow	Roles, review loop, PDF
14:30	Baseline checkpoint	Everyone tags "baseline" and pushes. Anyone behind gets help.
14:35 to 15:45	Phase 2: creative	Improve your app, resubmit, climb the board
15:45	Submissions close	The leaderboard freezes
15:45 to 16:30	Checks and open mic	Organisers check the top apps live. Everyone else shares "what broke and how I fixed it" and fills the feedback form.
about 16:45	Results	Top 3 announced
16:55	Group photo	
17:00	Close	

If Phase 1 runs late, Phase 2 is shortened. The baseline checkpoint is never cut.

10. The levels (Phase 1)

Each level has a short live demo, a copy-paste prompt from the prompt pack (specforge-prompt.json), about 20 minutes of building and a push to GitHub. Start every prompt with the stack header from the prompt pack.

Level	Time	Goal	Done when
L0 Spawn	11:35 to 11:55	Open your tool, start from the starter repo, link GitHub	First push done, repo link submitted
L1 Reader	12:00 to 12:25	Upload a spec and blueprint and extract them into structured data	The screen shows the CO list and units
L2 Template	12:25 to 12:45	Pick the Lab or Non-Lab template automatically, with a manual override, and export the filled document as spec.html	Right template, and spec.html downloads
L3 Mapper	12:45 to 13:40	AI CO-PO-PSO map (3/2/1) with a one-line justification per cell. Build POST /api/v1/generate, run npm run verify and submit both files.	Map table with justifications, and a score on the board
L4 Inspector	13:40 to 14:05	Validation report listing defects, saved in defects[]	The report shows issues on the sample
L5 Workflow	14:05 to 14:30	Author, Reviewer, HOD roles, DRAFT watermark, print/PDF, API endpoints	Reviewer cannot edit; HOD approves. This is your baseline.
Phase 2	14:35 to 15:45	Creative improvements that raise your score	Final submission before 15:45

Phase 2 is solo. Mentors can give ideas but never type into your project. Ideas that raise the score: better extraction prompts, a deterministic Bloom's verb lookup before the AI, a self-check pass where the model reviews its own map against the rules, more validation rules, and testing on a non-lab or spec-only case you create yourself.

11. Rules

1. Individual event. Each person builds and submits their own app, created from the event starter repo, in their own public GitHub repository or as a .zip of the project (see page 9).
2. Any AI tool is allowed. Disclose every tool in your README. You are responsible for every line the AI writes.
3. No hard-coding. Your output must be produced by your app from the inputs. Course text, CO lists or mappings written into the source code will be flagged, and the organisers decide the outcome.
4. Work must be done on the event day. Commits and zips after the 15:45 close are ignored.
5. Keep your API key secret. Enter it only in your app's Settings panel. Never paste it into a prompt, the README or a commit.
6. Use the stack in Section 5.
7. The input documents are for this event only. Do not publish them outside your repository.
8. Mentors answer questions about the problem, the documents and the academic rules. They do not write your code.
9. Be kind and professional. The organisers' decisions are final.

12. Results

- The ranking is the score from running your code after 15:45 (see section 8).
- Ties are broken by who reached their best upload score first, then by fewer submissions.
- Organisers also check the code for hard-coded course content, and look at the top apps live (roles, workflow, PDF). Hard-coding or an app that does not work can be disqualified.
- E-certificates are confirmed. The organisers announce results at the end of the day.

13. Getting help

- Stuck? Raise your hand or walk to the help desk. A mentor comes to you within about 5 minutes.
- Mentors work in zones around the room and rotate every hour.
- The help desk at the front handles GitHub sync problems and AI tool key or limit errors.
- The domain corner has faculty for CO-PO mapping and Bloom's questions.

14. Before you arrive

- Bring a laptop, charger, college ID and, if you have one, an extension cord.
- Create a GitHub account if you do not have one, and remember your login.
- Get one free AI key before you arrive: Gemini (aistudio.google.com) or OpenRouter (openrouter.ai). Either works; get both if you can. Never share your key.
- If you plan to code in your own editor (Cursor, VS Code with Copilot, or Gemini CLI), have it installed along with Git and Node.js 20 or newer.
- Create your repository from the starter repo: github.com/voxmastery/specforge-starter, press Use this template, keep it public.

How to get a free AI key (about 2 minutes)

You only need one of the two. If you get stuck, the help desk will help you get a key on the day. Keep your key private: anyone who has it can use your quota. If it leaks, delete it on the same page and create a new one.

Option 1: Gemini (Google AI Studio)

- Go to aistudio.google.com and sign in with a Google account.
- Click "Get API key" in the left menu, then "Create API key".
- Copy the key and keep it somewhere safe. You will paste it into your app's Settings box.
- Model name: use a Flash model from the model list in AI Studio.

Option 2: OpenRouter

- Go to openrouter.ai and sign up (Google or GitHub login works).
- Open your account menu, choose "Keys", then "Create Key". Copy it.
- Open the Models page and filter for free models. Free model names end in ":free". Copy one name for your app's Settings box.
- Free models have usage limits. If you hit one, switch to a different free model or to Gemini.

Handing in a .zip instead of a repository

No GitHub repository, or it will not push? Zip your project folder and upload the zip in Submit. If the zip has outputs/ inside (run npm run verify first), the zip alone is a full submission: both files are read from it. At 15:45 the organisers run whichever is newer: your latest zip or your last commit.

Put in the zip

- package.json, package-lock.json, server/, src/, scripts/, config/, templates/
- index.html, tsconfig.json, vite.config.ts and your README.md
- outputs/specforge_output.json and outputs/spec.html (these are scored)
- Anything else your app needs to run

Leave out

- node_modules/ (the organisers install packages, and it makes the zip too big)
- .env (it holds your API key; a zip with one is refused)
- .git/ and dist/. kit/ and sample-inputs/ are optional.

How to zip

- Windows: open the project folder, select everything except node_modules and .env, right-click, Send to, Compressed (zipped) folder.
- Mac: select the same files in Finder, right-click, Compress.
- Terminal (Mac, Linux, Git Bash), in the project folder:
  `zip -r specforge.zip . -x "node_modules/*" ".git/*" "dist/*" ".env*"`
- Up to 4 MB; the starter zips to about 0.3 MB. Each new zip replaces the last one. Zips can be uploaded until 15:45.