**Minecraft NeoForge 1.21.1 Modding Mastery  
College‑Style Course Syllabus & Detailed Outline**

*Prepared for: Self‑Paced / Udemy or Self‑Hosted Delivery*

*Version: 1.0 (October 30, 2025)*

# Course Overview

This course transforms a beginner modder with professional software experience into a full NeoForge mod developer. It follows a scaffolded, college‑style structure in four semesters: Foundations, Interactivity & Systems, World & Entities, and Professional Development. Each module introduces core concepts, then immediately applies them through labs and a cumulative project. By the end, learners publish a polished mod with GUIs, networking, entities, world generation, datapacks/configs, testing, and CI/CD.

## Learning Outcomes

* Set up and maintain a robust NeoForge 1.21.1 development environment.
* Design and register content: items, blocks, recipes, loot tables, tags, models, and blockstates.
* Implement stateful logic with Block Entities, Menus/Screens (GUIs), inventories, NBT persistence, and renderers.
* Use events, ticking, and custom recipe systems to implement gameplay mechanics.
* Build networking layers (packets) for client↔server sync and user interactions.
* Integrate fluids, energy/capabilities, and datapack‑driven balancing with reload listeners.
* Author worldgen features (ores, trees, structures, biomes) and custom entities with AI and attributes.
* Create villager professions/trades and add immersive audio, particles, and custom models.
* Ensure compatibility (tags, JEI), performance, multiplayer correctness, and automated tests.
* Package, license, and publish a production‑quality mod with CI/CD and documentation.

## Target Audience & Prerequisites

* Audience: Developers new to Minecraft modding (comfortable with Java/Gradle, OOP, and Git).
* Prereqs: Java 17/21 fluency, IntelliJ basics, Git/GitHub, command line familiarity.

## Required Software & Resources

* JDK 21 (or compatible with NeoForge toolchain).
* IntelliJ IDEA Community or Ultimate.
* Gradle wrapper (from MDK).
* NeoForge MDK for Minecraft 1.21.1.
* Git & GitHub account for version control and CI/CD.
* Optional: Image editor for textures (Aseprite/GIMP), sound editor (Audacity).

# Delivery & Assessment Model

* Delivery: Video lectures + written labs + reference code.
* Assessments: Lesson quizzes (formative), module labs (summative), semester capstones (major grade).
* Cumulative Capstone: Publish a complete mod on Modrinth/CurseForge with README, license, and CI build.

## Grading Breakdown (suggested for academic settings)

* Quizzes (per lesson): 10%
* Labs (end of each module): 35%
* Semester Capstone Projects (x4): 40% (10% each)
* Final Publication & Presentation: 15%

## Academic Integrity & Collaboration Policy

* Code sharing allowed for labs with attribution; copying without understanding is discouraged.
* Capstone project must be primarily the student’s work; external libraries and APIs allowed with credit.
* Follow licensing rules for assets and third‑party code.

# Four‑Semester Structure (Recommended Pacing)

Each semester groups thematically related modules. Self‑paced learners can compress or expand schedules.

## SEMESTER 1 — Foundations of Modding (Beginner → Intermediate)

### Module 1 — Environment & Setup

**Lessons:**

1. Intro to NeoForge & project anatomy (MDK, @Mod, event buses).
2. Install JDK/Gradle/IntelliJ; clone MDK and runClient/runServer/runData.
3. Mappings (official/Parchment) and build.gradle essentials.
4. Logging, configuration separation, and jar packaging.
5. Source control: repo layout, .gitignore, commits, branching.

**Lab / Project:**

Create ‘Hello NeoForge’ mod that logs on startup and exports a signed JAR.

**Learning Outcomes:**

* Run a clean dev environment reliably.
* Understand lifecycle entry points and run configs.

Estimated Duration: 4–6 hours

### Module 2 — Core Content: Items & Blocks

**Lessons:**

1. Deferred registers: Items, Blocks, RegistryObject lifecycle.
2. Assets: models, blockstates, lang files; resource folder structure.
3. Recipes & loot tables; shaped/unshaped, smelting, smithing.
4. Datagen basics: generate models, blockstates, loot, recipes.
5. Tags & creative tabs; vanilla derivatives (stairs, slabs, fences).

**Lab / Project:**

Ore Expansion Pack: custom ore → ingot pipeline with tags, recipes, and datagen.

**Learning Outcomes:**

* Create content fully via registries + JSON/datagen.
* Organize assets for maintainability.

Estimated Duration: 6–8 hours

### Module 3 — Tools, Armor & Materials

**Lessons:**

1. Tool tiers & custom behavior; hammer/paxel patterns.
2. Armor materials, trims, effects on hit and full‑set bonuses.
3. Horse armor and item properties/durability overrides.

**Lab / Project:**

Forged Tools set with full armor, special hit effects, and trims.

**Learning Outcomes:**

* Author tools/armor with tuned stats and effects.
* Leverage item properties for behavior.

Estimated Duration: 5–7 hours

### Module 4 — Items With Behavior

**Lessons:**

1. Food & status effects; contextual tooltips.
2. Custom fuel & burn times; intro to components/properties.
3. Event primer: subscribing safely; client vs server considerations.

**Lab / Project:**

Mystic Produce: edible items with randomized buffs and rich tooltips.

**Learning Outcomes:**

* Wire items into effects and events cleanly.
* Write safe, side‑aware handlers.

Estimated Duration: 4–6 hours

### Module 5 — Reading the Codebase

**Lessons:**

1. Navigating vanilla source; common classes & patterns.
2. Reverse‑engineering a furnace: inventory, smelting, fuel logic.
3. Survey of open‑source mods and best practices.

**Lab / Project:**

Create a reference doc mapping common tasks → vanilla classes/methods.

**Learning Outcomes:**

* Find exemplars quickly in vanilla and community code.
* Adopt patterns that age well across versions.

Estimated Duration: 3–4 hours

## SEMESTER 2 — Interactivity & Systems (Intermediate)

### Module 6 — Block Entities & GUIs

**Lessons:**

1. BlockEntity lifecycle: NBT save/load, server tickers.
2. Menus (containers) and Screens: inventory wiring and rendering.
3. ItemStackHandler and slot rules; syncing with DataSlots/packets.
4. Renderer basics (block entity renderer).

**Lab / Project:**

Infuser v1: ticking BE with input/output inventory and a basic GUI.

**Learning Outcomes:**

* Combine server BE logic with client GUIs.
* Persist and sync state correctly.

Estimated Duration: 6–8 hours

### Module 7 — Networking & Sync

**Lessons:**

1. SimpleChannel setup; protocol versioning.
2. Client→Server actions (buttons/keybinds) and server validation.
3. Syncing custom fields and partial updates; thread enqueuing.

**Lab / Project:**

Infuser v2: GUI button triggers server craft; state updates flow back.

**Learning Outcomes:**

* Design safe, minimal packet flows.
* Avoid side‑leaks and race conditions.

Estimated Duration: 4–6 hours

### Module 8 — Events, Recipes & Effects

**Lessons:**

1. Event bus deep dive: tick, interaction, entity events.
2. Custom recipe types & serializers.
3. Client effects: particles/sounds wired to gameplay events.

**Lab / Project:**

Daily Ritual: time‑based mechanic triggered by world ticks.

**Learning Outcomes:**

* Implement deterministic time/event mechanics.
* Author custom crafting rules.

Estimated Duration: 5–7 hours

### Module 9 — Fluids & Energy (Capabilities)

**Lessons:**

1. Fluid registration, tanks, rendering; fluid IO rules.
2. Capabilities: custom energy system; storage, generation, consumption.
3. Interplay: BE with inventory + fluid + energy.

**Lab / Project:**

Essence Generator: converts a fluid fuel into stored energy over time.

**Learning Outcomes:**

* Model multi‑resource systems cleanly.
* Expose capabilities for inter‑block interactions.

Estimated Duration: 6–8 hours

### Module 10 — Configs & Datapacks

**Lessons:**

1. ForgeConfigSpec: client/common/server; hot‑reload limits.
2. Datapack reload listeners; codecs and JSON schemas.
3. Balancing systems without recompiles.

**Lab / Project:**

Make Infuser/Generator values datapack‑driven with server config multipliers.

**Learning Outcomes:**

* Separate content from code for tuning.
* Respect server admin control surfaces.

Estimated Duration: 4–6 hours

## SEMESTER 3 — World, Entities & Immersion (Advanced)

### Module 11 — World Generation

**Lessons:**

1. Placed/Configured features; biome modifiers; JSON data flow.
2. Ore, trees, flowers, geodes; structure templates & jigsaws.
3. Custom biomes and integration with vanilla generation.

**Lab / Project:**

Crystal Caves: add ore veins, a small ruin structure, and vegetation.

**Learning Outcomes:**

* Author and register world features end‑to‑end.
* Debug placement and density issues.

Estimated Duration: 6–9 hours

### Module 12 — Entities & AI

**Lessons:**

1. Entity registration, models, renderers.
2. Goals/pathfinding; attributes and spawn rules.
3. Tameable/rideable/throwable entities; inventories; dyeable elements; render layers.

**Lab / Project:**

Forest Guardian: tameable mob with custom armor and support AI goals.

**Learning Outcomes:**

* Implement full‑featured mobs with custom visuals.
* Balance spawn and behavior for gameplay.

Estimated Duration: 6–9 hours

### Module 13 — Villagers, Trades & Economy

**Lessons:**

1. Villager professions, POIs, houses.
2. Trade definitions and progression.
3. Decorative content: paintings, models; in‑game currency patterns.

**Lab / Project:**

Elarion Merchant: new profession with curated, tag‑aware trades.

**Learning Outcomes:**

* Integrate with village systems safely.
* Design sustainable trade loops.

Estimated Duration: 4–6 hours

### Module 14 — Audio, Particles & Models

**Lessons:**

1. Sound events & block sound types; music discs.
2. Particles and spawn conditions.
3. 3D item models; basic animation workflows (optional Geckolib).

**Lab / Project:**

Enchanted Workshop: animated station with particles and soundscape.

**Learning Outcomes:**

* Create immersive feedback for mechanics.
* Ship polished audiovisual content.

Estimated Duration: 4–6 hours

## SEMESTER 4 — Professional Mod Development (Expert)

### Module 15 — Inter‑Mod Compatibility & Integrations

**Lessons:**

1. Inter‑op via tags and capabilities; safe optional dependencies.
2. JEI/REI categories and recipe displays.
3. Public API design for your mod.

**Lab / Project:**

Expose Infuser recipes to JEI; publish a tiny public API class.

**Learning Outcomes:**

* Play nicely with the mod ecosystem.
* Stabilize your own extension points.

Estimated Duration: 4–6 hours

### Module 16 — Optimization, Testing & Multiplayer

**Lessons:**

1. Tick budgets, caching, and lazy evaluation.
2. Thread safety: enqueuing to main thread, avoiding deadlocks.
3. Automated tests (GameTest), integration tests, and MP sync audits.

**Lab / Project:**

Stress‑test worlds + GameTests for Infuser/Generator; MP desync checklist.

**Learning Outcomes:**

* Ship performant, reliable code.
* Prevent regressions with tests.

Estimated Duration: 5–7 hours

### Module 17 — UX, Accessibility & Localization

**Lessons:**

1. HUD overlays, in‑game guides, contextual tooltips.
2. Localization workflows and accessibility considerations.
3. Config GUIs and onboarding.

**Lab / Project:**

Add a HUD meter, localized strings, and an onboarding tooltip flow.

**Learning Outcomes:**

* Improve player comprehension and reach.
* Offer quality‑of‑life surfaces.

Estimated Duration: 3–5 hours

### Module 18 — Packaging, Licensing & Publishing

**Lessons:**

1. Metadata (mods.toml), semantic versioning, changelogs.
2. Licensing assets and code; attribution.
3. CI/CD with GitHub Actions; Modrinth/CurseForge publishing.

**Lab / Project:**

Publish a release build with README, license, gallery, and CI pipeline.

**Learning Outcomes:**

* Distribute professionally with confidence.
* Communicate changes transparently.

Estimated Duration: 3–5 hours

# Capstone Projects & Suggested Pacing

* Semester 1 Capstone: Content Pack – cohesive items/blocks/recipes with datagen and tags.
* Semester 2 Capstone: Systemic Machine – BE + GUI + networking + events + config.
* Semester 3 Capstone: Living World – worldgen feature + custom entity + trades.
* Semester 4 Capstone: Production Release – polished mod with tests and CI, published.

## Recommended Weekly Schedule (Example 16‑Week Term)

1. Week 1: Module 1
2. Week 2–3: Module 2
3. Week 4: Module 3
4. Week 5: Module 4
5. Week 6: Module 5 (Capstone S1 planning)
6. Week 7: Module 6
7. Week 8: Modules 7–8
8. Week 9: Module 9
9. Week 10: Module 10 (Capstone S2 delivery)
10. Week 11: Module 11
11. Week 12: Module 12
12. Week 13: Module 13
13. Week 14: Module 14 (Capstone S3 delivery)
14. Week 15: Modules 15–16
15. Week 16: Modules 17–18 (Final Release & Presentation)

# Assessment Rubrics (Summaries)

## Lab Rubric (10 pts)

* Correctness (4): meets functional requirements, no crashes.
* Code Quality (3): clear structure, naming, side separation (client/server).
* Assets & UX (2): correct assets, helpful tooltips/feedback.
* Git Hygiene (1): meaningful commits; README updates.

## Capstone Rubric (20 pts)

* Scope & Integration (6): combines required systems coherently.
* Stability & Performance (5): no dupes/leaks; smooth in MP.
* UX & Polish (4): GUI clarity, localization, sound/particles where appropriate.
* Docs & Release (3): README, changelog, license, screenshots.
* Testing (2): GameTests or reproducible test scenarios.

# Policies & Best Practices

* Client/Server Separation: never reference client‑only classes from common/server code; use Dist guards.
* Threading: handle networking on the correct thread; enqueue to main thread for world changes.
* Data Ownership: keep authoritative state server‑side; sync minimal diffs to clients.
* Versioning: tag releases; keep CHANGELOG; avoid breaking API without major version bump.
* Licensing: respect Mojang EULA; license your code/assets; credit third‑party works.

# Appendix A — Example Repository Layout

/src/main/java/com/example/mod  
 ModMain.java  
 registry/ (Blocks, Items, Menus, BlockEntityTypes, Sounds, Entities)  
 content/block/... content/item/... content/be/... client/screen/...  
 network/ (SimpleChannel, packets)  
 world/ (features, biomes, structures)  
 data/ (loaders, codecs, reload listeners)  
/src/main/resources/  
 meta/mods.toml  
 assets/<modid>/{lang,models,blockstates,textures,particles,sounds}  
 data/<modid>/{recipes,loot\_tables,tags,worldgen}

# Appendix B — Checklists

GUI Checklist

* MenuType registered; factory opens on block use; slots wired.
* Screen registered on client; texture path correct; tooltip overlays.
* Sync: DataSlots or packet; server validates actions.
* Close handling: drop items or persist; handle chunk unloads.

Networking Checklist

* SimpleChannel with protocol version; packet IDs centralized.
* Handlers enqueue to main thread; side checks; minimal payload.
* Partial sync where possible; rate‑limit noisy updates.

Worldgen Checklist

* Configured + Placed features registered; biome modifiers applied.
* Datapack JSON validated; density balanced; debug placement in dev world.
* Structures have templates, processors, and jigsaw rules.