Forsk Coding School

Powered Learning for Future Success

COURSE BROCHURE / JAIPUR

FULL STACK Development with Node.js



Build React interfaces.
Create Node.js APIs.
Connect SQL and NoSQL data.

FROM THE INTERFACE TO THE DATABASE

Build the whole app. Connect every layer.

01

TRAIN

Frontend, Node.js, SQL and NoSQL

02

INTERN

Full-time internship at Forsk Technologies

03

INTERVIEW

Preparation, feedback and placement support

FULL-TIME INTERNSHIP WITH

Forsk Technologies
From AI to AGI – We Build What's Next

YOUR FULL STACK LEARNING PATH

Build the interface.
Connect the whole system.

Progress from Web Design and React to Node.js APIs, databases and a working full stack application.

FRONTEND

01
Web Design
HTML, CSS and layouts

02
JavaScript
Core, ES6 to ES10 and APIs

03
React + Redux
Components, hooks and RTK

NODE.JS BACKEND

04
Runtime & async
Modules, files and HTTP

05
Express APIs
Routes and application logic

06
Access & validation
Accounts and permissions

DATA & DELIVERY

07
SQL & NoSQL
Complimentary training

08
Integration & tests
Connect and check the stack

09
Project & deployment
Build, publish and explain

WHO CAN START HERE

Beginners can start with Web Design and JavaScript. Learners joining the Node.js module directly should be comfortable with JavaScript and asynchronous code.

YOUR PORTFOLIO DIRECTION

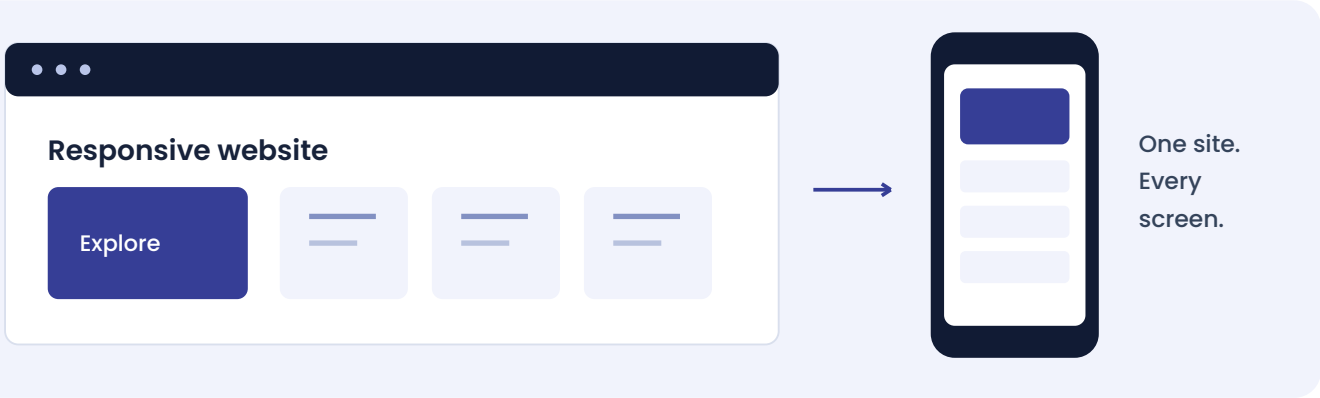
Build an application you can explain from end to end.

Develop skills relevant to Full Stack, Node.js Backend and React Developer roles.

MODULE 01 / WEB DESIGN

Build the structure.
Design for every screen.

Learn HTML, CSS, Bootstrap, Tailwind CSS and Core JavaScript. Turn a page layout into a responsive, interactive website.



HTML foundations

Semantic pages, links, images, lists and forms.

CSS layouts

Selectors, the box model, Flexbox and Grid.

Bootstrap

Responsive grids, components and utilities.

Tailwind CSS

Utility classes, spacing and responsive styling.

Core JavaScript

Variables, conditions, loops, functions and DOM events.

Usable interfaces

Readable layouts, keyboard access and form feedback.

PRACTICE PROJECT

A responsive business website or portfolio

Build navigation, content sections and an interactive enquiry-form interface.

MODULE 01 / LAYOUT & USABILITY

Make layouts adapt.
Keep every control usable.

Go beyond a single screen. Practise page structure, flexible layouts and clear interactions, then apply Bootstrap and Tailwind CSS.

01 **Structure with meaning**

Use headings, landmarks, links, images and labelled form fields. Choose elements that match the content and interaction.

02 **Control the CSS**

Work with selectors, inheritance, specificity, spacing, typography and the box model. Inspect styles in browser developer tools.

03 **Build flexible layouts**

Combine Flexbox, Grid, relative sizing and media queries. Check content wrapping, navigation and images at different viewport widths.

04 **Make the page usable**

Practise keyboard navigation, visible focus, useful image descriptions and clear form messages. Check readable text and colour contrast.

TWO STYLING APPROACHES

Bootstrap

Practise its grid, spacing utilities, forms and reusable components. Customise a consistent page layout.

Tailwind CSS

Compose interfaces with utility classes. Practise responsive variants, interactive states and consistent design values.

LAYOUT LAB

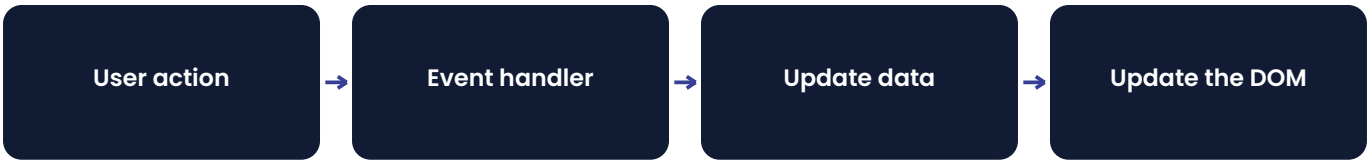
Rebuild one layout in two approaches.

Create a responsive landing page, then compare its Bootstrap and Tailwind implementations.

MODULE 01 / CORE JAVASCRIPT

Turn static pages into working interfaces.

Build a programming foundation before moving into React. Connect values, conditions and functions to real browser interactions.



Values & decisions

Variables, strings, numbers, booleans, operators and comparisons. Use conditions to decide what an interface should do.

Loops & functions

Repeat work with loops, create functions with parameters and return values, and understand scope and reusable logic.

Arrays & objects

Model lists and records. Read, add, update and remove values; practise transforming data for a page.

DOM & events

Select elements, update content and classes, and handle clicks, input and form submission through event listeners.

Forms & browser storage

Validate input, display helpful messages and practise localStorage for non-sensitive preferences. Work with JSON data.

Debugging habits

Use console output, breakpoints and the call stack. Reproduce an error, inspect values and explain the correction.

JAVASCRIPT LAB

Build a task list with filters and saved preferences.

Create, complete and remove tasks; validate input and explain each event handler.

MODULE 02 / ADVANCED JAVASCRIPT

Add behaviour.
Work confidently with data.

Build on Core JavaScript with ES6 to ES10, reusable modules, asynchronous code and API-driven interfaces.

01 **Modern JavaScript**

Let and const, arrow functions, template literals, destructuring, spread syntax and modules.

02 **Arrays, objects & logic**

Transform collections, work with objects and JSON, and organise reusable functions.

03 **Asynchronous programming**

Promises, async/await and Fetch for requesting and processing application data.

04 **Reliable user experiences**

DOM interactions, error handling, loading and empty states, browser tools and debugging.

MAKE EVERY STATE CLEAR

IDLE

Waiting for user input

LOADING

Request is in progress

SUCCESS

Show the returned data

ERROR

Explain what went wrong

PRACTICE PROJECT

An API-powered search interface

Fetch results, filter or sort data and handle loading, empty and error states.

MODULE 02 / API INTEGRATION

Request the data.
Handle every outcome.

Connect a frontend to an existing or practice API. Understand the request, inspect the response and keep the interface useful when conditions change.

01 / REQUESTS

HTTP & JSON

Practise URLs, query parameters, methods, headers and JSON payloads. Inspect requests and response data in the browser network panel.

02 / ASYNC FLOW

Promises & async/await

Follow asynchronous execution, consume Fetch responses and handle failures. Check HTTP status as well as network errors.

03 / RESPONSE HANDLING

Use the result carefully

Parse JSON, check the expected data shape and render results. Practise cancelling outdated requests when user input changes.

04 / USER EXPERIENCE

Keep feedback clear

Show loading, empty, success and error states. Offer a useful retry path and avoid displaying stale results as current data.

PRACTISE CHANGING CONDITIONS

Try a slow response, no matches, a failed request and repeated searches.

API LAB

Build a searchable catalogue from a practice API.

Add filters and pagination where the API supports them; show clear loading and error feedback.

MODULE 03 / REACT.JS

Think in components. Build interactive apps.

Break an interface into reusable components, pass data through props and respond to user actions with state and hooks.

01 / COMPOSITION

Components & JSX

Create reusable components, pass props and render conditions and lists with suitable keys.

02 / INTERACTION

State & hooks

Handle events and practise useState, useEffect, refs and reusable custom hooks.

03 / USER INPUT

Forms & validation

Manage controlled inputs, validate values and give clear feedback during form submission.

04 / APPLICATIONS

Routing & API integration

Organise page navigation, request data and handle loading, empty and error states.

THE REACT INTERACTION LOOP

User event → State update → Render → Updated UI

Practise a searchable product listing or task board using reusable React components.

MODULE 03 / REACT IN PRACTICE

Choose the right state.
Keep components clear.

Use hooks with a purpose. Separate user interactions, derived values and external synchronisation so application behaviour stays understandable.

State updates

Practise useState with objects and arrays. Use an updater function when a new value depends on the previous value.

Derived values

Calculate filtered lists and totals from props or state. Avoid storing duplicate values that can fall out of sync.

Effects & cleanup

Use useEffect for external synchronisation. Understand dependencies and clean up listeners, timers or requests when appropriate.

Refs & focus

Use useRef for DOM access and values that do not drive rendering. Practise focusing an input after an interaction.

Custom hooks

Extract reusable behaviour into custom hooks. Give the hook clear inputs and outputs and keep components focused on their interface.

Where state belongs

Keep temporary state close to its use. Lift shared state to a common parent and practise Context for wider access.

REACT LAB

Build a filterable dashboard with reusable hooks.

Explain which values are stored, which are derived and which interactions need an effect.

MODULE 03 / COMPLETE USER JOURNEYS

Make inputs reliable.
Make navigation clear.

Connect screens and interactions with React Router and form handling. Practise flows that still make sense when input is invalid or a URL is unknown.

FORMS & VALIDATION

Capture input

Controlled text fields, selects and checkboxes; field values and change handlers.

Validate clearly

Required values, formats and helpful field messages. Preserve useful input when a submission fails.

Handle submission

Pending, success and failure feedback. Prevent accidental repeated submissions and manage focus.

ROUTING & PAGE FLOWS

Map the screens

Create routes and shared layouts. Connect pages with links and meaningful navigation.

Use URL information

Read route parameters and query strings for detail pages, search terms and filters.

Handle the edges

Add a not-found view. Check direct links, refresh, back navigation and nested page layouts.

EXAMPLE ROUTES

/products /products/:productId Not-found page

USER-JOURNEY LAB

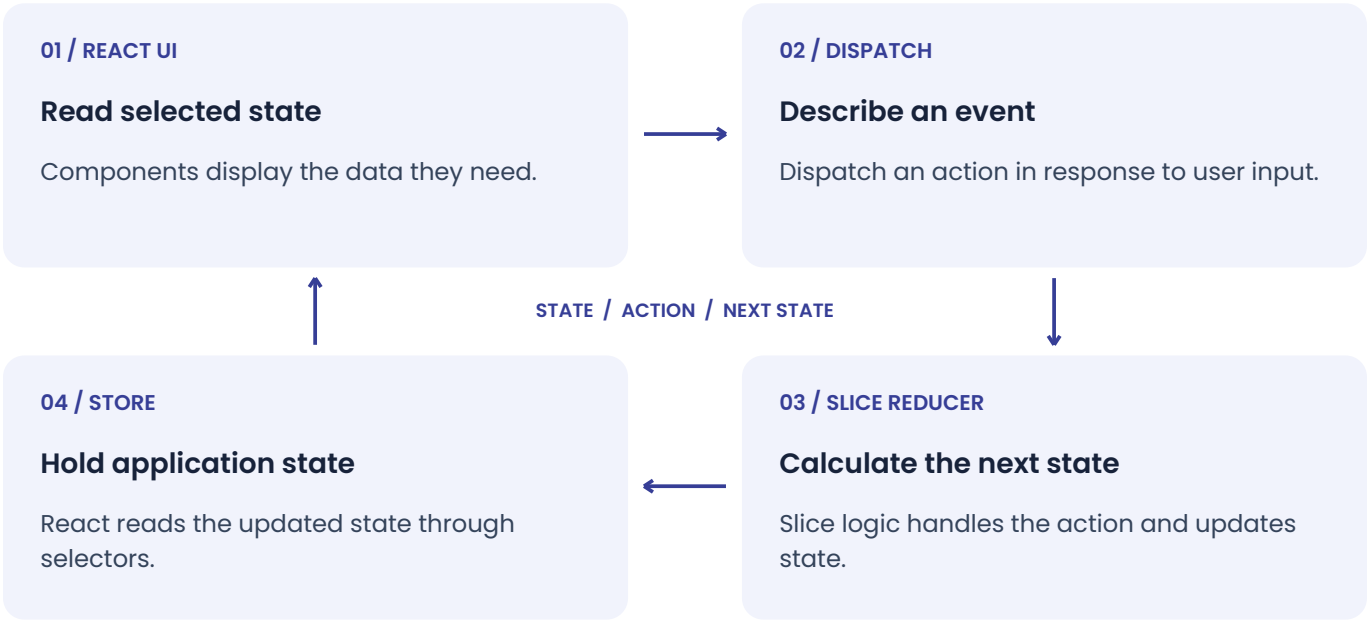
Build a catalogue, detail page and enquiry form.

Review validation, route changes and failure states using a practice API or mock data.

MODULE 03 / REDUX + RTK

Make shared state predictable.

Use Redux and Redux Toolkit (RTK) to organise shared application state and connect state changes to React interfaces.



Store & slices

configureStore, createSlice and action creators.

React-Redux hooks

useSelector, useDispatch and derived UI state.

Async workflows

Async thunks and pending, success and error states.

API state with RTK Query

Query and mutation basics, caching and refetching.

PRACTICE PROJECT

A shared cart or task-management feature

Keep items, quantities, filters and totals consistent across multiple screens.

MODULE 03 / APPLICATION DATA

Know where each piece of data belongs.

Distinguish local interaction state, shared client state and data from a server. Choose a clear approach for each part of the application.

LOCAL UI

An open dialog or an unfinished input usually belongs near the component that uses it.

SHARED CLIENT STATE

A cart or shared selection can live in a Redux slice and be read through selectors.

REMOTE DATA

An API catalogue needs fetching, loading feedback and a strategy for keeping cached results current.

RTK Query

Define API queries and mutations; use generated hooks to read request state. Practise caching, refetching and refreshing affected data after a change.

Async workflows

Practise createAsyncThunk when a workflow needs custom async logic. Handle pending, fulfilled and rejected outcomes in the slice and UI.

INSPECT THE DATA FLOW

Use Redux DevTools to inspect actions and state. Keep business data serialisable and calculate derived values with selectors.

DATA LAB

Connect a shared cart to an API-backed catalogue.

Show how local UI, cart state, remote data and request failures are handled separately.

NODE.JS / SERVER-SIDE JAVASCRIPT

Take JavaScript beyond the browser.

Use Node.js to run application logic, handle requests and work with files and data. Build on the JavaScript foundations covered earlier.

Runtime & asynchronous work

Understand the event loop, callbacks, promises and async/await. Recognise blocking operations and handle asynchronous failures.

Modules & npm

Organise code with modules. Practise imports/exports, CommonJS concepts, package.json, scripts, dependencies and lockfiles.

Files & paths

Use file-system and path APIs to read and write data. Handle missing files and separate configuration from application logic.

Buffers & streams

Understand binary data and processing data in chunks. Explore streams for file handling and avoid loading large files unnecessarily.

HTTP fundamentals

Create a basic server. Read request information, set response headers and status codes, and return structured JSON.

Configuration & structure

Work with environment variables, separate development settings and organise application folders. Keep server secrets out of frontend bundles.

NODE.JS LAB

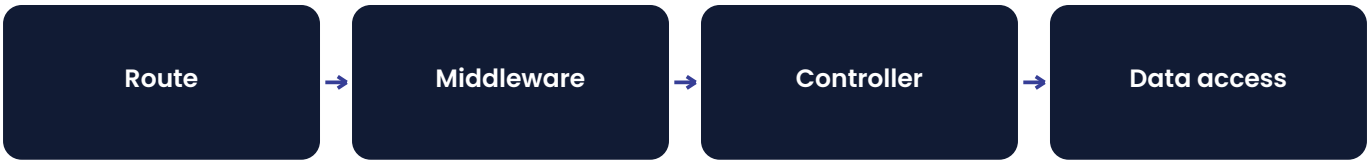
Create a small HTTP service with file-based input.

Explain the request, asynchronous work, response and failure handling before adding Express.

NODE.JS / EXPRESS APPLICATIONS

Give every request a clear path through the app.

Build REST-style APIs with Express. Separate request handling, validation, business logic and database access.



Routes & resources

Use HTTP methods, route parameters and query strings. Design consistent URLs for records, collections, filters and pagination.

Middleware

Handle shared work such as parsing, logging and access checks. Understand middleware order and when to pass control onward.

Input validation

Check request bodies, identifiers and query values on the server. Return useful validation messages and appropriate status codes.

Controllers & services

Keep route handlers focused. Place reusable business rules in services and isolate database operations behind clear functions.

Errors & responses

Use a consistent response shape, handle missing records and centralise error handling. Keep internal stack traces out of public responses.

Files & API documentation

Explore uploads with size/type limits. Document endpoints, request examples, response formats and error cases for frontend integration.

API LAB

Create, list, update and delete tasks through an API.

Add filters, validation and consistent responses; document successful and failed requests.

NODE.JS / ACCOUNTS & PERMISSIONS

Know who is signed in. Check what they can do.

Build account flows and server-side permissions into a practice application. Treat identity, authorisation and validation as separate responsibilities.

01 / IDENTITY

Register & sign in

Validate account input, store password hashes and check credentials. Return careful error messages and practise sign-out behaviour.

02 / AUTHENTICATION

Sessions & token concepts

Compare session-based and token-based flows, including JWT concepts. Practise expiry, verification and appropriate cookie settings.

03 / AUTHORISATION

Roles & record ownership

Check permissions on the server for every protected action. Distinguish a signed-in user from a user allowed to change a record.

04 / APPLICATION CONTROLS

Protect the request path

Practise HTTPS, validation, request limits and parameterised queries. Configure cross-origin access and protect server-side configuration.

CHECK ACCESS IN THE API

A hidden button in the browser does not enforce a server-side permission.

ACCOUNT LAB

Add member and admin actions to a task application.

Test expired sessions, invalid input and attempts to change another user's record.

INCLUDED / SQL DATABASES

Model the relationships. Query with confidence.

Complimentary SQL training is included with this Full Stack course. Cover relational foundations with MySQL, PostgreSQL and Oracle.

01 **Tables & relationships**

Define tables, data types, primary and foreign keys, constraints and relationships. Practise schema design and normalisation basics.

02 **Read & change data**

Use SELECT, INSERT, UPDATE and DELETE. Add filtering, sorting, joins, grouping and aggregate queries.

03 **Consistency & performance**

Explore transactions, commit/rollback and indexes. Understand how schema and query choices affect correctness and query work.

04 **Use SQL from Node.js**

Connect through a database driver or query layer. Practise connection pooling, parameterised queries and handling database failures.

SQL COVERAGE

MySQL

Tables, relationships and application queries.

PostgreSQL

Relational modelling, queries and transactions.

Oracle

Relational foundations and SQL query practice.

DATABASE LAB

Model customers, products and orders.

Write related queries and expose selected data through a validated Node.js API.

INCLUDED / NOSQL, SEARCH & ANALYTICS

Choose a data model that fits the workload.

Complimentary coverage includes MongoDB, Cassandra, DynamoDB, Elasticsearch and Druid. Explore their models and different application uses.

DOCUMENT DATA / MONGODB

Build with documents and collections.

Practise document structure, CRUD operations, filters, indexes and aggregation basics. Connect Node.js through a driver or an ODM and validate data before saving it.

Cassandra

Wide-column modelling, partition keys and designing tables around expected queries.

DynamoDB

Key-value and document models, partition/sort keys and access-pattern planning.

Elasticsearch

Search and analytics: indexes, text queries, filters and aggregations.

Apache Druid

Analytical data: ingestion, time-based queries and aggregate reporting.

Compare the query pattern, data shape and consistency needs before choosing a store for a feature.

DATA-MODELLING LAB

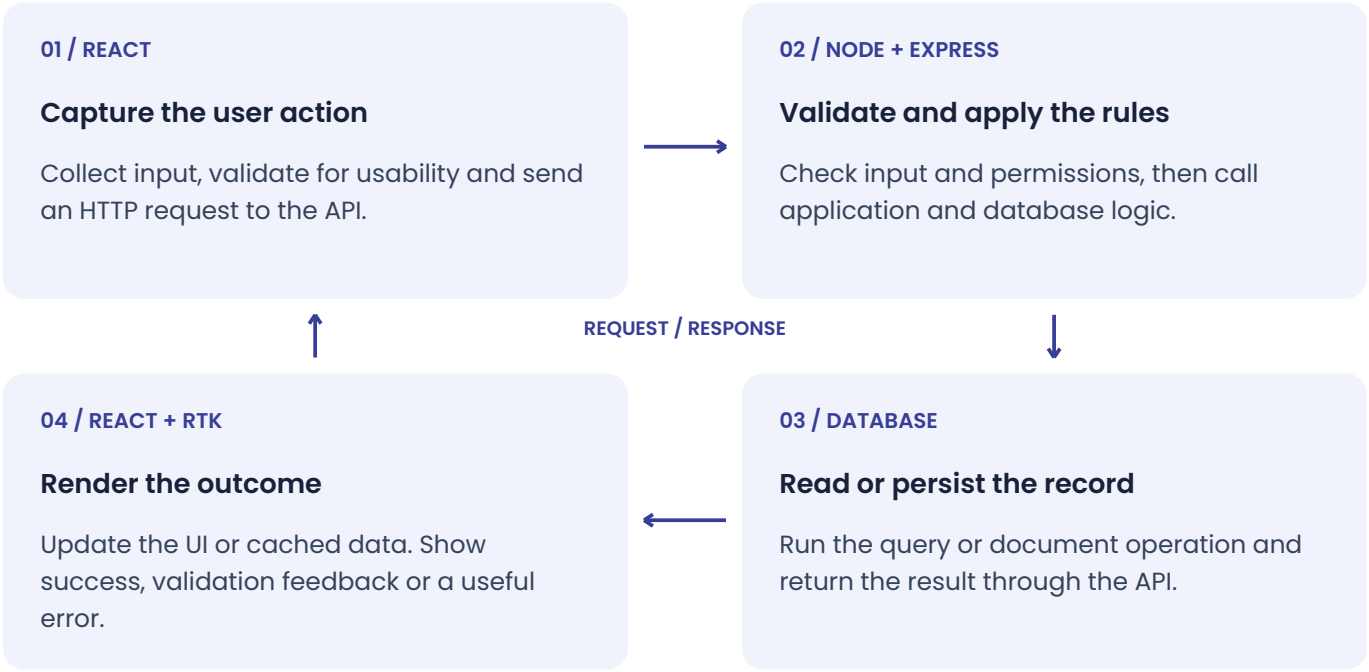
Build a document-backed catalogue API.

Explain when relational tables, documents, search indexes or analytical stores suit the feature.

FULL STACK / ONE WORKING FEATURE

Connect React to Node.
Bring the data back.

Join the frontend, API and database around a single user journey. Keep each layer responsible for a clear part of the application.



AGREE ON THE API CONTRACT

Align URLs, request fields, response shapes and error messages. Configure the frontend API address and cross-origin behaviour for the development and deployed environments.

INTEGRATION LAB

Create a task, save it and see it after refresh.

Demonstrate the complete journey, including invalid input, access checks and failed requests.

NODE.JS / VERIFY THE BEHAVIOUR

Test the expected path. Then test the edges.

Use repeatable checks to verify API behaviour and full stack journeys. Trace failures through the browser, server and database.

01 / LOGIC

Unit-level checks

Test small functions and validation rules with controlled inputs. Explore assertions and Node.js test-runner fundamentals.

02 / API

Endpoint checks

Verify methods, status codes and response bodies. Cover invalid payloads, missing records and unauthorised requests.

03 / INTEGRATION

Database-backed flows

Use a separate test database or controlled fixtures. Verify saved records, relationships and cleanup between test runs.

04 / DIAGNOSIS

Useful logs and errors

Trace a request across layers. Log useful context, inspect stack traces during development and keep credentials out of logs.

REPEATABLE PRACTICE

Keep request examples and test commands with the project so a reviewer can reproduce the checks.

TESTING LAB

Create a small API test collection and automated checks.

Demonstrate valid, invalid, unauthorised and missing-record cases with expected results.

PORTFOLIO PRACTICE

Build something you can demonstrate.

Choose a full stack project scenario and connect React screens, Node.js APIs and a database. Show a complete, explainable user journey.

01 Course enquiry portal

Course listings, saved enquiries and an admin view.

02 Catalogue & orders

Products, cart state, saved orders and status updates.

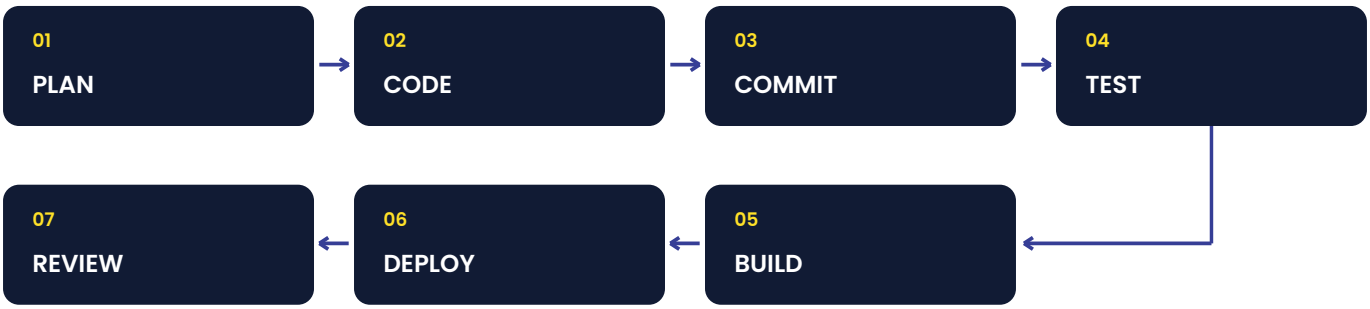
03 Team task board

Accounts, task ownership, status changes and filters.

04 Inventory dashboard

Items, stock records, search and aggregate reports.

FROM A FEATURE TO A DEPLOYED APPLICATION



Review screens, API validation, access checks, saved data and failure states.

CAPSTONE EVIDENCE

Make your work easy to review.

Include a repository, setup guide, API examples, database setup and a working demo.

CAPSTONE / BUILD, CHECK & EXPLAIN

Make the project work.
Make it easy to review.

Turn a chosen project into a complete full stack demo. Review the frontend, Node.js service and database together before presenting it.

REVIEW AREA	EVIDENCE TO DEMONSTRATE
Frontend journey	Check responsive screens, keyboard access, form feedback, navigation and loading, empty and error states.
API & access	Review server-side validation, account flows, permissions, response formats and repeatable endpoint checks.
Database setup	Document schema or collection setup, sample data and configuration. Check queries, relationships and persisted changes.
Configuration	Separate frontend public settings from server secrets. Configure API addresses, cross-origin access and database connections.
Build & deployment	Build the React app, start the Node.js service and check HTTPS, routes, assets, logs and database connectivity.
Handoff & Git	Provide useful commits, setup commands, API examples, test instructions, known limitations and a repeatable demo.

PROJECT HANDOFF

A demo, a repository and a clear explanation.

Present the problem, key decisions, test cases, limitations and the improvements you would make next.

COURSE FEES / INDIAN RUPEES

Frontend + Node.js.
One Full Stack course.

Frontend training is ₹20,000 and Node.js Backend is ₹8,500. SQL and NoSQL database training is complimentary with this Full Stack course.

COURSE MODULE	FEE (INR)
01 **Web Design** — HTML, CSS, Bootstrap, Tailwind & Core JavaScript	₹7,500
02 **Advanced JavaScript** — ES6 to ES10	₹4,000
03 **React + Redux + RTK** — React applications and Redux Toolkit	₹8,500
04 **Node.js Backend** — Node.js, Express APIs and full stack integration	₹8,500

TOTAL FULL STACK COURSE FEE

Frontend ₹20,000 + Node.js ₹8,500

₹28,500

COMPLIMENTARY / SQL + NOSQL

SQL: MySQL, PostgreSQL, Oracle
NoSQL, search & analytics: MongoDB, Cassandra, DynamoDB, Elasticsearch, Druid

TRAINING TO FULL-TIME INTERNSHIP

Learn. Build.
Then put it into practice.

Your journey begins with training at Forsk Coding School and continues with a full-time internship at Forsk Technologies.

01 / TRAINING INSTITUTE

Forsk Coding School
Powered Learning for Future Success

Learn Full Stack with Node.js through structured course content, practical assignments and project work. Build the technical foundation you will carry into your internship.

AFTER TRAINING

02 / FULL-TIME INTERNSHIP

Forsk Technologies
From AI to AGI – We Build What's Next

Apply your learning during a full-time internship at Forsk Technologies. Develop practical experience and prepare to explain your skills and project work in company interviews.

WHAT COMES NEXT

Company preparation and interviews.

Assessment, feedback and focused preparation continue across interview opportunities.

CAREER PREPARATION

Prepare. Interview. Improve. Repeat.

Company interviews may be remote or on site. Each round informs the preparation for the next opportunity.

01 / PREPARE

Company-specific preparation

Revise skills, practise questions and explain your project work.

02 / INTERVIEW

Meet the company

Attend the company's interview, remotely or on site.

FEEDBACK INTO PREPARATION

03 / ASSESS

Review the interview

Assess performance, discuss feedback and identify skill gaps.

04 / PREPARE AGAIN

Work on the gaps

Practise the weak areas and prepare for the next company.

When selected: progress through the company's offer and joining process. Selection depends on the employer's requirements and interview outcome.

SUPPORT THROUGH THE PROCESS

Resume & GitHub portfolio • Mock interviews • Project presentations

START YOUR FULL STACK JOURNEY

7231968183

forskcodingschool.com

COURSE DETAILS