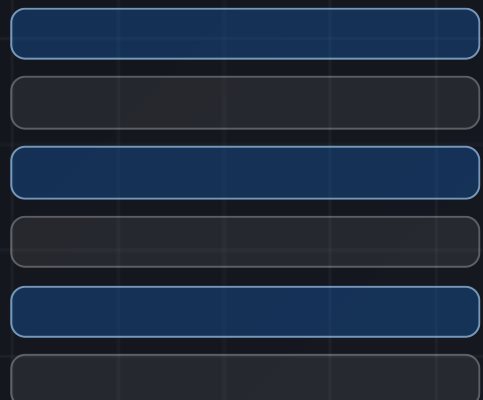


WHITE PAPER

Beyond Prompt-to-Code: Intended Design

Why AI needs a checked Intended Design of the whole application, and how the engine turns it into code.



CONTENTS

What is inside

01	Executive summary	one page
02	The problem	why software stays hard, and why AI sharpens it
03	The approach	six views, four properties
04	The Intended Design	a connected graph, and the shape behind it
05	One language for logic	typed, close to C#
06	The engine and its targets	knowledge that executes
07	AI and the Intended Design	why the graph is the right context
08	The tools	Studio, zAI, the engine
09	Ownership and architecture	what you get, and what you keep
10	Adopting it	roles, a learning path, a first hour
11	Glossary and status	one term per concept

Reading guide: decision-makers will get most from sections 1, 2, 3, 7 and 9. Developers will want 4, 5, 6 and 8.

Software should be described once, and produced many times

zAppDev is a browser-based platform that connects natural language AI to software engineering. You describe a business application with AI, and the description, an Intended Design, is checked across database, server and screens before any code is produced. The engine then produces ordinary source code for your target stack, which you can download and run anywhere. Every service can become an MCP tool, automatically. People and AI agents work on the same checked Intended Design of the application.

AI is here to stay. Today an agent infers your architecture from your code, helped by memory, skills and specs that are advice. Give it the architecture itself, verified as a whole, and check every change against it before any code exists.

What it is

The approach describes an application across six connected views: Intent, Domain, Logic, UX/UI, Integration and Data. Every element in every view knows what it is related to, so the whole Intended Design forms a **connected graph**. Behaviour is written in one language. **Studio** is the browser workspace where all of it is created, tested and built.

What it changes

- **Code stops being the source of truth.** The validated Intended Design is. The engine turns it into a complete, readable application for a chosen target, today .NET and Angular.
- **Engineering knowledge becomes executable.** The engine runs a program for each target, an **implementation strategy**, that encodes patterns, architecture, libraries and security practice. It runs with no inference or further reasoning and gives the same result every time, however large the application.
- **AI gets the right context.** Instead of guessing at relationships in thousands of lines of code, an AI works on a connected, validated, much smaller Intended Design, and its changes are checked before anything is generated.
- **You can take the result with you.** The produced code is real, readable and portable. Download it, and the Intended Design files.

Who it is for

Teams that build and maintain line-of-business applications and want them consistent and governable. Architects who want standards enforced by programs, not reviews. Developers who want to spend their time on the interesting work. Organisations adopting AI who want it to work inside guardrails.

Software stays slow, fragile and locked away

Four problems are familiar to every team that builds software for a living.

Rebuilt again and again

Every new stack means starting over. What the application does is trapped inside the code of the last one.

Disconnected parts

Data, rules, screens and APIs drift apart. A change in one place breaks another, and nobody sees it coming.

Platforms you cannot leave

The tool becomes the runtime. Stop paying and you stop having an application.

Knowledge that does not scale

Good practice lives in wikis, guidelines and people's heads. It has to be read, understood and applied again by every person, every time, and it drifts.

AI sharpens the problem

Language models have made code cheap to produce. They have not made it cheap to keep coherent. Asked to change a large system from a few files, an LLM guesses at the relationships it cannot see, and produces plausible code with no way of proving it is right. The faster code is produced, the faster inconsistency accumulates.

This changes where the value sits. When code is abundant, the scarce things are a precise statement of what the application is, and a way to check changes against all of it. Both live above the code.

WITHOUT AN INTENDED DESIGN

Prompt → AI → Raw code → Hope

WITH AN INTENDED DESIGN

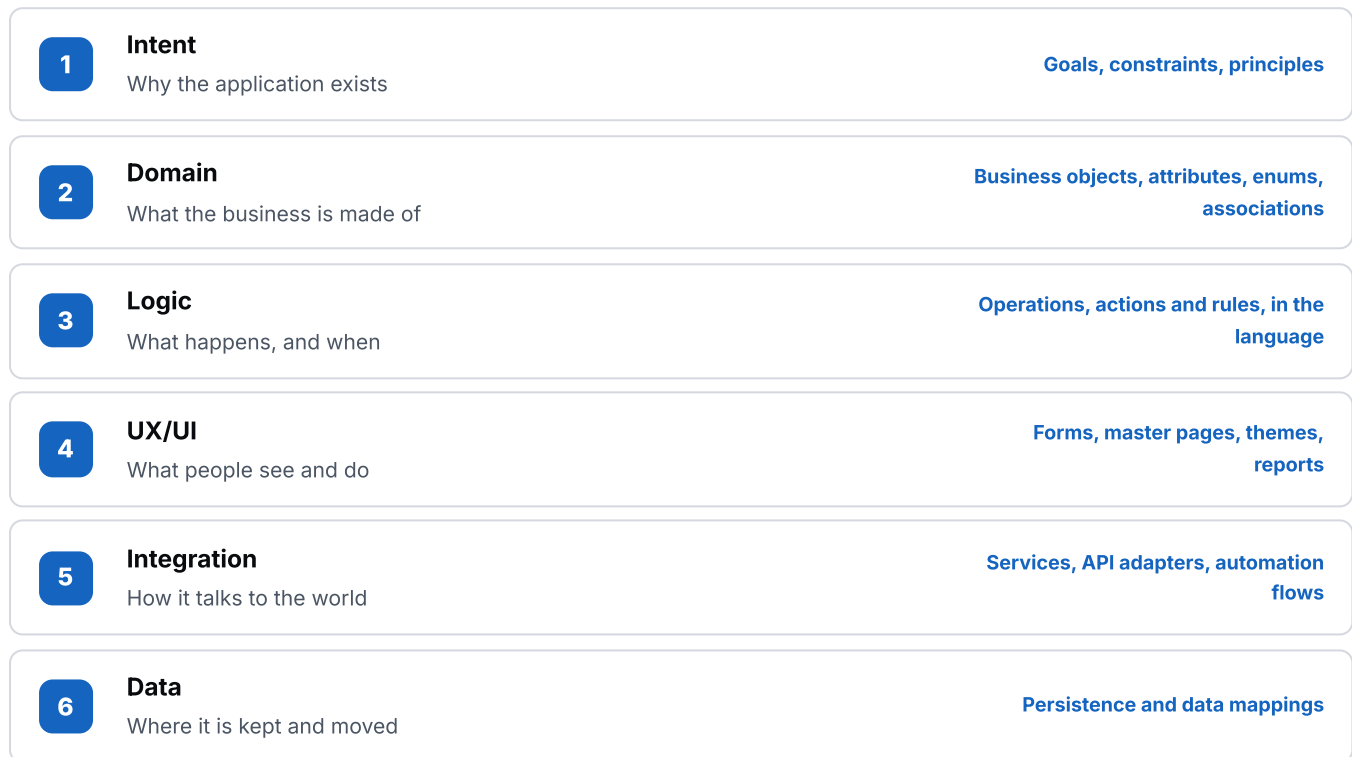
Prompt → AI reads the graph → Validated Intended Design → The engine →

Application

Describe the architecture in views

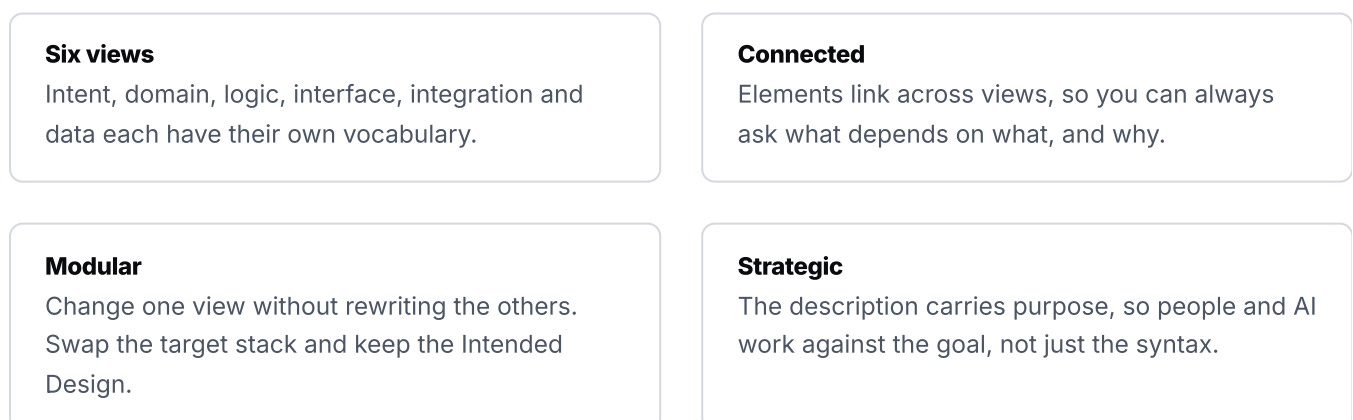
For fifty years we have written code and hoped the intent survived inside it. This approach reverses that. You state what the application is, at each level of abstraction, and the code is produced from that statement.

Six connected views



Each view has its own vocabulary. Intent describes why the application exists: its goals, constraints and principles.

Four properties



Why it fits how people think

People do not hold an application in their heads as files. They hold purposes, things, rules and screens, and the links between them. Six views keep each question small, and links mirror how ideas connect. A tool with the same shape makes understanding and description line up.

An application as a connected graph

Every element is a node and every relationship is an edge. The Intended Design is a graph you can ask questions of.

The **Intended Design** is the complete description of one application: structure drawn in Studio and behaviour written in the language. It is the single source of truth. Ask what would break if a Role object changed, and the graph answers across every view, because the relationships are explicit rather than implied.

The shape of every Intended Design

Behind every application is the same shape: the types of element zAppDev knows, and the named relationships allowed between them. It is what the checks enforce, and what an AI learns once. The next page shows it.

What an element type holds

ELEMENT TYPE	MAIN CONCEPTS
Business Object	Classes, attributes, enums, associations (with multiplicity, navigability and delete behaviour), operations
Form	Controls and layout, view model, controller actions with permissions, datasets; WebForm, MasterPage or PartialView
Service	Operations (verb, path, access), data contracts; REST or MCP
API Adapter	An external system: operations, structs; REST, SOAP, library, database and others
Automation flow	Steps with logic and conditions, an optional schedule, inputs and outputs
Data Mapping	Left and right structures joined by named, field-level mappings
Report	Pages, headers and footers, a view model, entry points
Theme	Design tokens, variations such as light and dark, component styles, custom LESS
Unit Test	Test operations and helpers, with initialise and clean-up
Configuration	Roles, permissions and settings

The element types, in six views, joined by named relationships. Solid blue arrows are confirmed in the platform's own files; dashed amber ones are proposed and still to be confirmed.



Typed, close to C#

Behaviour is written once, in a clear, typed language that lives inside the Intended Design.

The language for the logic of the architecture has a syntax and type system close to C#, and it is used wherever logic lives: class operations, form actions, service operations, automation steps and tests. Because it is part of the Intended Design, every statement knows which objects, forms and services it uses, and it is checked against all of them.

WHAT YOU WRITE (THE LANGUAGE)

```
function Domain.InteractionDTO CreateInteraction(
  Domain.InteractionDTO data) {
  var owner = Domain.PMOUser.GetAll()
    .First(o => o.Roles.Any(
      r => r.Name == Application.Roles.SalesMan));
  int accountId;
  if data.AccountId > 0
  {
    accountId = data.AccountId;
  }
  else
  {
    accountId = data.Account.CreateAccount().Id;
  }
  var lead = Domain.Lead.Init(
    data.FlowId, accountId, data.Title, owner);
  data.UpdateLead(lead);
  lead.Save();
  return Domain.InteractionDTO.Fill(lead);
}
```

WHAT IS GENERATED (C#)

```
public MyApp.B0.InteractionDTO
CreateInteractionImplementation(
  MyApp.B0.InteractionDTO data) {
  MyApp.B0.PMOUser owner = new MyApp.DAL.Repository()
    .GetAsQueryable<MyApp.B0.PMOUser>()
    ?.FirstOrDefault((o) => o.Roles.Any(
      (r) => r.Name == "SalesMan"));
  int? accountId = new int?();
  if ((data?.AccountId ?? 0) > 0) {
    accountId = (data?.AccountId ?? 0);
  }
  else {
    accountId = (data?.Account?.CreateAccount()?.Id
    ?? 0);
  }
  MyApp.B0.Lead lead = MyApp.B0.LeadExtensions.Init(
    (data?.FlowId ?? 0), accountId, (data?.Title ??
    ""), owner);
  data?.UpdateLead(lead);
  new MyApp.DAL.Repository().Save<MyApp.B0.Lead>
    (lead);
  return
  MyApp.B0.InteractionDTOExtensions.Fill(lead);
}
```

Excerpt from a service operation in a customer-relationship application, and the C# generated from it. Names are anonymised and indentation is added for width.

What this gives you

- **One language, many targets.** The same logic can be produced for a different stack, because it is not written in that stack's language.
- **Validation across layers.** A reference to a missing object or a wrong type is caught in the Intended Design, before any code exists.
- **Traceability.** A code map links every generated method back to the Intended Design element it came from, so a stack trace leads to its source.

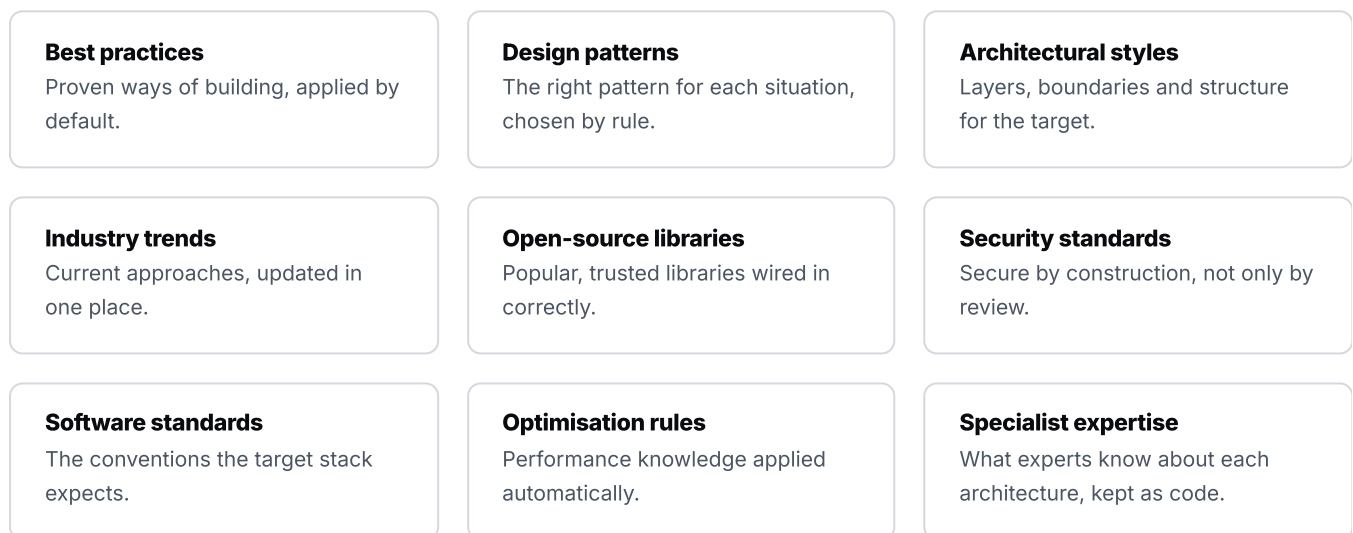
Knowledge that executes

For each target, the engine runs a program, an implementation strategy, that encodes how great software is built, and applies it with no inference or further reasoning, on applications of any size.

Most engineering knowledge lives in documents. It has to be read, interpreted and applied by each person or each AI, every time. The engine encodes that knowledge as code, so it is applied consistently everywhere.



What the engine encodes



Why a program beats a document

- **Executed.** It executes the rules. Nothing is read, planned or reasoned about at generation time, however large the application.
- **Without guessing.** Every decision is a rule, not an interpretation. The same Intended Design always produces the same code.
- **Consistent at scale.** The ten-thousandth file follows the same patterns and standards as the first.
- **Improved once.** When a practice, library or standard moves on, the knowledge is updated in the engine, not hunted down across many code bases.
- **Kept current, as a service.** Each target is maintained so that it follows the latest versions of the frameworks and libraries an application is built on (for example .NET, Angular, PrimeNG, NHibernate and the PDF and Excel libraries). To move forward you run a build, with no reasoning or inference involved. Your own logic and custom libraries remain yours to keep current, and your tests run on every build.

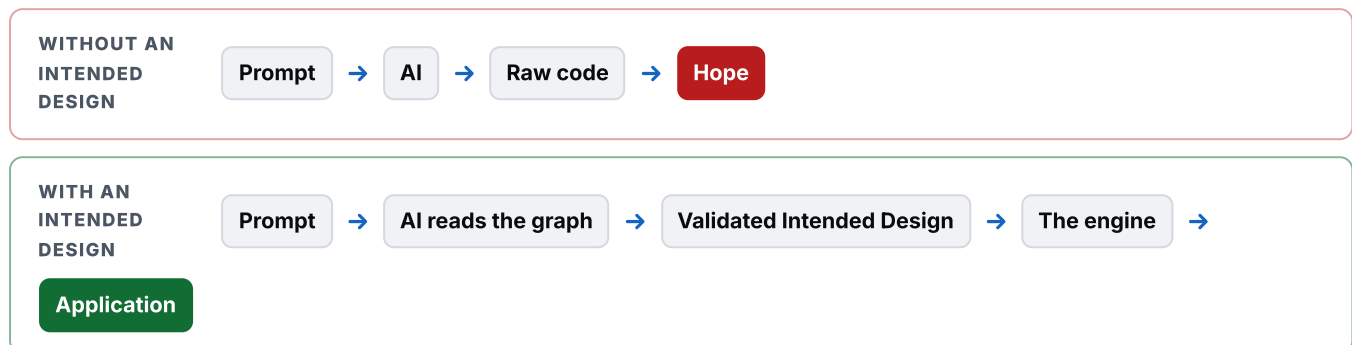
Write your own

Because a target is produced by a program, it is not limited to the ones that ship. An organisation, or a specialist, can encode its own architecture, standards and expertise, and the whole Intended Design benefits at once, with no rework of the Intended Design.

Availability of tooling for writing custom targets is described in the documentation and may change.

Why the graph is the right context

If an LLM can write code directly, why describe the application first? Because scale, alignment and speed all depend on what the LLM is given to read.



Four reasons

1. The whole system in view

Asked to add an approval workflow for high-budget items, an LLM working from raw code inspects a few files and guesses. Working from the graph, it sees every entity, relation, cascade rule and security flag at once, in a compact form, without thousands of lines of boilerplate.

2. Rules that are stated, not guessed

"Make BudgetItem depend on Budget" can be built several ways in code. In the Intended Design it is one explicit relationship with its multiplicity and delete behaviour. The AI does not choose a pattern. It translates a specification.

3. A guardrail before any code exists

A language compiler checks types within code. Validating the Intended Design checks relationships across the whole application: a broken link, a wrong type, a rule that contradicts another. The failure message is precise, so the AI can correct itself immediately.

4. One description, any target

Because the graph describes the application and not one language, the same change can be produced for any target.

Two stages

In stage one the AI changes the Intended Design, quickly, because there is little to process. In stage two the engine produces the same code every time from the validated Intended Design. There is no AI call in that loop, so the same Intended Design always gives the same application.

The self-correction loop

AI proposes a change. The checks report exactly what is wrong. The AI reads the message and fixes it. Only a valid Intended Design reaches the engine. Mistakes cost seconds, not a failed release.

Open to your own agents

zAI inside Studio is one way in. Any agent that speaks the Model Context Protocol (MCP) can use the same Intended Design through tools to read the application, add or change classes, attributes and associations, validate the Intended Design and build, under the same checks that protect a person working in Studio.

Studio, zAI and the engine

Studio is the browser workspace where the Intended Design is created, tested and built. It runs in the browser, with nothing to install.

Ten editors, one workspace

EDITOR	WHAT YOU DO THERE
Business objects	Draw classes, attributes and associations as a diagram. Persistence and API follow.
Forms	Lay out screens visually, bind them to your objects, add rules and validation.
Logic	Write behaviour in the language, with completion and feedback as you write.
Services	Expose operations as REST or MCP, set the verb, path and access, implement each in the language.
API adapters	Describe an external system once: operations, parameters and structures.
Automation flows	Build work that runs without a person, as steps and logic, with a schedule.
Data mappings	Map one shape of data to another, field by field.
Reports	Design printable documents visually, bound to your data.
Themes	Set the look with design tokens, light and dark variations, and styles.
Unit tests	Write tests in the language and run them in Studio before every build.

zAI

zAI works inside Studio on the Intended Design itself. You describe a change in plain language, zAI reads the relevant part of the graph and proposes it, the Intended Design is validated, and you review it like any other change.

The engine

The engine works in three phases.

1 · Graph validation

The whole Intended Design is checked: missing links, type mismatches, broken references.

2 · Optimisation

The graph is simplified and normalised before anything is written.

3 · Production for a target

The chosen target produces the code, the database and the interface for its stack.

Starting points

New applications begin from a template: **Empty** (the theme only) or **Web application** (sign-in, users, roles, permissions, settings, a Welcome page). Two live demonstration applications, Lite ERP and Issue Manager, are available to open and explore.

What you get, and what you keep

No black box in the way: the output is a standard application you can read, build and host yourself.

What is produced

CONCERN	OUTPUT
Server	.NET: controllers, view models, business layer and data access
Client	Angular and TypeScript: pages, components and routing
Data	A schema generated from your business objects
Interfaces	Exposed REST services, MCP tools and the adapters you described
Security	Forms authentication, roles and per-action permissions enforced on the server, with a configurable password policy
Quality	Unit tests you write in Studio, run in Studio and in your pipeline
Localisation	Resource files with every user-facing text

Framework versions follow the current release; check the documentation for exact versions.

What you keep

- **The code.** It is real and readable. Open it in Visual Studio, put it in your own Git repository, run it on your own servers.
- **The Intended Design.** It is the portable description of what you built, independent of any one stack.
- **The freedom to move.** Download the generated solution and the Intended Design files at any time.

Governing at the right level

Reviews, approvals and audits can happen on the Intended Design, where intent is visible, while the generated code stays open for anyone who wants to read it. The engine enforces standards as programs, so compliance is a property of the build, not of a review.

From first Intended Design to leading the build

Intended Design, describing an application so that people and machines build the right thing, is a skill that can be learned in stages, by building.

Three ways in

Designer

You know the business. Describe domains, screens and rules, and watch them become a working application.

Design developer

You write code. Learn the language, extend the Intended Design where it needs precision, and open the generated solution in your own tools.

Design architect

You shape systems. Think in views, choose targets and govern change at the level of intent.

A learning path

LEVEL	YOU CAN	STATUS
1 · Describe	Describe objects and associations	AVAILABLE
2 · Compose	Build forms, lists and navigation	AVAILABLE
3 · Behave	Add logic in the language	AVAILABLE
4 · Connect	Use services, adapters and MCP	AVAILABLE
5 · Govern	Work with intent, review and standards	COMING SOON
6 · Lead	Set targets across teams and stacks	COMING SOON

Your first hour

1. **Create a free account** and open Studio in your browser.
2. **Pick the Web application template.** Sign-in, users, roles and a Welcome page are already there.
3. **Ask zAI for a Customer list.** Review the change, build, and see it run.

Describe it. Check it. Run it.

Create a free account at ide.zappdev.io, and read the full documentation on the documentation site.

One term per concept

zAppDev

The application development platform.

Studio

The browser workspace where applications are described, tested and built.

Checked Intended Design

A description of the whole application, checked mechanically by the engine, from which the code is produced. An AI works from it instead of inferring the application from code.

View

One level of abstraction: Intent, Domain, Logic, UX/UI, Integration or Data.

Intended Design

The complete description of an application: structure drawn in Studio and behaviour written in the language.

Element / Relationship

A node in the Intended Design, and a link between two elements, within or across views.

Connected graph

The Intended Design seen as connected elements and relationships.

Shape

The types of element and the named relationships allowed between them: the shape every Intended Design follows.

The language

The typed language for the logic of the architecture, close to C#.

The engine

Checks and optimises the Intended Design, then produces the application for the chosen target.

Target

One stack the engine produces for, through an implementation strategy: a program that encodes the practices, patterns, architecture, libraries and security knowledge for one target and executes it with no inference or further reasoning, and consistently.

zAI

The AI assistant inside Studio that proposes changes to the Intended Design.

MCP

Model Context Protocol: lets AI agents use an application through standard tools.

Template / Sample

A starting point for a new application, and a complete example application to learn from.

What is available, and what is proposed

CAPABILITY	STATUS
Forms, logic, services, adapters, flows, mappings, reports, themes, tests	AVAILABLE IN STUDIO
Studio editors, the language, validation, version control, deployment	AVAILABLE
Generation of .NET and Angular applications	AVAILABLE
MCP server for every application, and zAI	AVAILABLE
Live demo applications: Lite ERP, Issue Manager	AVAILABLE
React Native (Expo) mobile target: client only, REST GET and POST	PREVIEW
More demos: POS, Charts, Swagger	PLANNED

This paper describes the platform as of October 2026. Details that change, such as versions and plans, are in the documentation.

© 2026 zAppDev. All rights reserved. zAppDev is the name of the zAppDev platform. This document is provided for information and may be shared in its original, unmodified form. Other product and company names are the property of their respective owners.