Tutorial: How To Go From a Domain Model to Fully Deployed Microservice in 60 Seconds, with Parallel Agile CodeBot™

Parallel Agile CodeBot™ is an add-in for Enterprise Architect that generates a complete REST API from your UML domain model. The API is written in Node Express.js with a MongoDB database. CodeBot also creates Swagger-based documentation, plus client libraries in several programming languages to access your new API. Not bad for a few seconds’ work!

The generated API can be used as the basis for rapid, feedback-driven prototyping following a domain modeling workshop. To top it all, CodeBot automatically hosts your prototype API on a secure cloud service, so you don’t actually need to set up a server or database… just run the generator, and a few seconds later you’re trying out an executable version of your domain model with its own live database.

In this article we’ll walk through how CodeBot can be put to practical use during domain driven prototyping, to help give your project a solid foundation on which to build. Domain driven prototyping is a highly effective process with code automation at its core, and forms a major part of the Parallel Agile software development process.

You can actually try out CodeBot for free via a 30-day license (including the cloud API hosting): just head on over to https://parallelagile.net to get started.
So, to set the scene…

You have a project that’s just getting started, with multiple developers (perhaps even multiple teams) set to work on different parts of the new system.

You need a way for everyone involved to work in parallel with as little communication overhead as possible. The key, of course, is to create a detailed domain model, well-known these days as the ubiquitous language, which then kind of permeates the entire project.

The project kicks off with a domain modeling workshop, from which you’ll generate the first iteration of the working prototype. The workshop should involve all the project stakeholders: the customer, business analyst, architect, UX designer, and the senior developers. But for all of that, this initial workshop should also be kept short (like, an hour or two)… it’s really just a springboard to get the prototype started.

Prototyping itself is where you’ll discover the bulk of the domain details – new fields, objects, relationships, validation rules, behavior etc.

For the example in this article we’ll create a much-simplified car rental system. Don’t expect to put this particular system into production, we’ll use it purely to illustrate the process!

**Early Version of the Domain Model**

During that initial short workshop, a model begins to emerge:

Well, it’s a start. Car Rental domain model, fit the first...

The team hasn’t yet defined any attributes or behaviors, which usually suggests that not enough thought has yet gone into the model: the team’s collective brain hasn’t deep-dived
into the business domain yet – and (just to emphasize this point) a lot of that detail will emerge later during prototyping.

Actual system behavior, in the form of use cases and UX workflows would be defined soon after domain modeling, but not during... you kind of want the model definitions to settle down somewhat before you start using them as building blocks.

(Note this might seem like it differs slightly from traditional Domain Driven Design, where the domain model definition includes system behavior. We use the same definition, just a slightly different approach to reach the same goal).

So, about that first Car Rental diagram…

The relationships between the domain objects are a fairly typical first stab at joining the dots. They do make sense conceptually: a business person can relate to:

\[
\text{customer RENTS vehicle WITH rental agreement}
\]

However, these kinds of relationships don’t map especially well to an API and database schema. In terms of data, there isn’t a physical relationship pointing from Vehicle to Rental Agreement: a particular vehicle would never “know” about a rental. On the other hand, a Rental Agreement certainly would have a reference to a vehicle (“what exactly is being rented here?”). So, from an “entity relationship” viewpoint the arrow would really be reversed.

After a coffee break, the team returns to the meeting room (the one with the nice panoramic view and the potpourri in the corner) and digs deep. Finally, the following model rolls out of the workshop (your own model will contain many more domain objects of course, we’re just keeping this to a small example here):
As you can see, there’s still a long way to go, but we now have some attributes, and some more thought has been put into the relationships, particularly around multiplicity (“Can the customer have more than one address?” etc). The idea is to add as much detail as possible into the domain model. The richer it is, the more precisely it represents the business domain, and the more details CodeBot will have to chew on.

But even in this early, raw state, the domain model provides something that you can generate and benefit from. Let’s do that now...

**Running Parallel Agile CodeBot™ For the First Time**

When you run CodeBot, at a broad level it creates two things: a zipfile containing all of the project source code, API documentation etc; and a secure, deployed copy of the API up in the cloud, which you can start testing straightaway.

There aren’t very many steps involved to run CodeBot, and to be honest they’re pretty self-explanatory. But just to illustrate, here’s what you need to do...

Optionally, create a new project via the blue + over on the right (see the next screenshot); or alternatively just use the Default project that’s automatically created:
A unique code is created from the project name (e.g. “Car Rental Example” will be turned into “carrentalexample”)\(^1\), and this in turn is used in the URL for the hosted API, and other places.

As you can see in the panel on the right, this project doesn’t contain or do very much yet. Let’s do something about that now…

To run the CodeBot add-in, switch over to your copy of EA and bring up your domain model:

---

\(^1\) You can change the project code if needed, e.g. if it’s generated an unfortunate portmanteau; just double-click the project code in Edit Project.
Right-click on the package or within the diagram, and choose:

**Specialize > Parallel Agile > Generate Project**

You’ll be asked to login, and then to choose which project to use for this model.

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th># Runs</th>
<th>Last Run</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car Rental Example</td>
<td>Example project to d</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>TikTok Go</td>
<td>3D VR/AR game with a</td>
<td>7</td>
<td>26 April 2019</td>
</tr>
<tr>
<td>Default project</td>
<td>This is the default</td>
<td>16</td>
<td>22 April 2019</td>
</tr>
</tbody>
</table>
```

The add-in will ask you to choose where to save the new zipfile once it’s generated. Then it kicks into action:
Once it’s finished developing your domain-driven project, you’ll see the “All done” screen, similar to the following (click the “Get token” button to see the connection details):

The details shown in the above screenshot enable you to connect to your new cloud-hosted API. All of the connection details that you see in the screen are also available on the website; more about that in a second though.

The add-in has also downloaded and saved the zipfile containing your generated project. Click the “Open folder” button and extract the files…
The next screenshot shows a subset of what’s been created. Please note the Server folder, which contains the Node Express API code, isn’t available via the Free license; however you do still get a free, cloud-hosted deployment of the API that you can test, plus all the generated client code for connecting to the hosted API, etc:

Let’s take a look inside one of the generated files…
This next screenshot shows the top of JavaScript/Adapter.js – this is part of the JS-based REST client, which you can use in your own client code to access the generated API. As you can see, this is already set up to call the hosted API directly.

```javascript
var url = "https://parallelagile.net/hosted/mattco/carrentalexample/Dev/";

// If your API service is running locally,
// replace this with: http://127.0.0.1:2000/carrentalexample/

var DBAdapter = {};

DBAdapter.create = function(collection, data, successCB, errorCB) {
    if (isDefined(collection) && isDefined(data) && isArray(data) // isObject(data)
        xhrCall(collection, "POST", "", data, successCB, errorCB);
    } else {
        errorCB("Error: " + "Invalid Parameters");
    }
};

DBAdapter.get = function(collection, param, successCB, errorCB) {
    // ...
};
```

Note: For convenience, an access token has been generated and placed in the generated client file. To keep your API secure, this token lasts for 24 hours; you can generate a new token anytime via the web console.

The Server code is all set up so that you can run it as a Node Express.js REST-based microservice. It can also be deployed without modification as a “serverless” AWS Lambda function (it achieves this via the open source serverless-http library) – in fact this is what we’re using “behind the scenes” to automatically create and deploy the cloud-hosted APIs.

A discussion of the possibilities opened up by having a serverless deployment target is probably outside the scope of this article, but it is a good example of how CodeBot can be used to generate a completely different deployment target, database, or even a different architecture, based on the same domain model.

Having generated the project, let’s flip back to the CodeBot web console now to see what’s new, and take your new API for a quick drive around the block...
Trying Out Your New Hosted Prototype API

Here’s the CodeBot web console, in which some new features have suddenly appeared:

We have lift-off! CodeBot has automatically deployed your new API into a cloud-hosted environment for testing. If you want to use Postman (or Curl, or some other REST client) to try out the hosted API, click “Connection details”.

Alternatively, you can test the API directly from the CodeBot web console, courtesy of the generated Swagger documentation. Click “API Docs”; this brings up the complete Swagger documentation for your API:
To test creating a database record via the API, open up one of the domain objects (we’ll choose Vehicle in the example) to see a list of the REST API endpoints available for this object, then open up the POST endpoint at the top, click **Try it now**, fill in some data (or use the generated defaults), scroll down and hit **Execute**:

![API endpoints screenshot](image1)

The first time the hosted API is called, it’ll take a good few seconds as it’s running from a cold start. Subsequent calls are quicker; after a period of inactivity it then goes back to sleep until it’s next prodded awake.

You should see a result similar to the next screenshot…

![Post request result screenshot](image2)
Result! One new domain record created in the cloud… you can also then run search queries and retrieve any data you’ve created via the GET endpoint.

As you can see, CodeBot can easily be run and re-run iteratively to provide instant feedback on what the business domain will look like when translated into a technical solution. This approach really comes into its own when you’re defining the system behavior (use cases, UX flows) based on the domain model. This stage would happen soon after the domain modeling sessions, with the customer back in the room.

It’s a good idea to circulate the hosted URL and screenshots to the team, to gather feedback. This feedback can then be collated ready for when you start to define the behavioral requirements. This also gives the UX designer time to mock up a rough UI, directly from the domain model and the generated prototype.

Talking of UIs, we’re currently improving CodeBot to also generate a UI for browsing the hosted data in something that looks more like a web application; so you would be able to show this to a non-technical customer as a quick demo of their domain model as a forms-based application.

Although CodeBot is genuinely useful for domain driven prototyping, it’s also designed to be the cornerstone of large-scale projects with multiple teams, enabling them to work in parallel with minimized cost of communication. Additionally, the executable architecture it generates is robust enough to be carried through to production – we’ve seen it used on several projects with excellent results.

Find out more about Parallel Agile at http://parallelagile.com, and try out CodeBot for free at https://parallelagile.net. And for the latest developments, be sure to follow our blog over at Medium.