JEREMY TAMMIK: Thank you for coming early in the morning. Lots of empty chairs. That's unplanned for. It's booked out. Poor people.

So my name is Jeremy Tammik, and my topic is exactly what it says here-- connect desktop and cloud, free your BIM data. And you've been hearing a lot in the past few days about Autodesk Forge, the Autodesk platform for achieving this goal. And it does fully, much more fully than what I'm discussing here. I have one sample which also includes Forge and has a special twist.

Because Forge is basically read only You have a CAD model. You translate it to much more compressed web streams, JSON data. You can view it. You can share it globally. And you have full access to all the data. So that's a good thing.

But it's read only. You can only translate from the CAD seed file to the web representation, and you can't get it back again. So my little twist is how can I change something in the web and get it back again. So that's the Forge part of this presentation. That's right at the end.

And basically, this freeing your BIM data-- my focus is not so much on putting the entire, huge set of CAD's model onto the cloud myself, because that's too hard. That's something Autodesk can do. And Forge does it, and that's brilliant and fantastic for those who need it.

But in many cases, to optimize a workflow, you can look carefully at what do I actually need. Maybe I don't need a terabyte of CAD data which ends up as maybe 100 kilobytes of really compressed web stream data. Maybe I need just one little piece of information which I want to make available everywhere, all the time, to anyone, or to somebody specific. It depends on your workflow.

So that's basically your task-- to decide what is my optimal workflow, what piece of data do I want to share, and how do I connect that. And the aim of this presentation is to show that's really easy-- very, very easy. And if it's hard for you, you're doing something wrong. And you just need to rethink and restart and find the easy way. There's an easy way.

Who here is familiar with the Revit API? That's good. And who is familiar with any kind of web programming? That's also very good. Cool. So you're well-prepared. That's great.

This is an overview of our goals. But basically, I've said it all already. Another aspect of this is with my BIM data, it's only valuable, really, or useful if I can address everybody who needs to know whatever it is I want to share. And especially in the BIM domain, this number can grow by orders of magnitude between different phases of the design, obviously.

So if I build an airport, I might have a couple of thousand people-- couple of hundred planning design, a couple of thousands, or tens of thousands, building, but actually millions later on who need access to this data in some way just to find the way. And all of that can be extracted from one single model. So that's very handy.

And that's why it's important to be able to make this BIM data globally accessible-- some of it, probably not all. So those are things you need to think about. I said this is easy. Everything I show here except Revit and the Forge translation is open source. So I'll show a couple of other examples, older ones which are not Forge-based. And they use 100% open source software.

And even in Forge, the viewing part of Forge is completely open source. The only proprietary step in Forge is the translation from the seed CAD model to the web streamable format plus some of the later translations to other file formats.

And I'll also dive in briefly into NoSQL databases. Who knows what the difference is between SQL and NoSQL? OK. Well, then I have some news for some of you guys. I picked this out specifically because it was so surprising to me when I discovered it in 2013. And it was really, really old back then.

So it's really really, really old back now. So if any one of you is working with SQL or with databases of any kind whatsoever and has not explored NoSQL databases, I would advise that should be the very first thing you do. Actually, that's more important than all the rest here, I think.

OK. So this is what I mentioned, but it's pretty obvious. This has a little bit to do with Forge, but basically more to do with why would I want to put any kind of data on the cloud. I want to put some data on the cloud because it's important to share it with people. And I probably don't want to share everything, but there's almost certainly something that I want to share. Almost any application has some little piece of information that wants to be shared, and that can make the whole system more efficient and make use of it properly.

And those key words that we see associated with Forge all the time-- to design, visualize, collaborate, make, and use. And especially in the BIM area, the number of people in these different phases vary hugely. Yeah. That's basically the same thing, so I'm talking about connecting Revit and BIM as in desktop in Forge.

This also has to do with the trends that we see that you're all aware of and have heard hundreds of times already-- the shrinking number of desktop computers and applications and use. People run around with phones and expect not to be tied to their desktop and still have access to the tools that they need from some specific workflow.

And I said this is almost all open source. And almost everything that you need is available in some shape or form in some library on the internet. So it really makes-- it is worthwhile exploring what is out there that already does what I need. I don't need to program much myself. What I end up programming is glue code to connect existing components, sometimes a small little piece of customization. In my Forge example at the end, it's based on Phillip [INAUDIBLE] really brilliant Forge sample here which does almost everything that Forge can do.

And the interesting thing is that my enhancement of it-- I won't go through it all now in detail-- consists of less. He's written probably 10,000 lines of code just to demonstrate everything. My enhancements to achieve this connection back to BIM consist of less than 25 lines of code.

That's your job-- to find what is it I want to achieve, how can I minimize it, simplify it. And then there's so little to do.

Security is a big issue, so I would advise you not to deal with it somehow. I don't. Forge does it for you. And that's a large part of the complexity. If you get started with Forge, you have to deal with all this two-legged authorization, three-legged authorization. Who knows the difference? OK.

In one sentence, two-legged is my application wants to access my or your data. Three-legged-- my application wants to access somebody else's data, and I need permission from this other person in between. So this other person must be involved for a second to give me the credentials. So two-legged can be totally transparent. I can hard code that, more or less. Three-legged always asks the person to say yes, OK, go for it.

But anyway, the different tools that I mentioned-- all of this open source stuff, or proprietary systems like Dropbox and so on, they have their own security systems. And if you trust them or set up your system that you don't break anything, then you can basically trust them as far as you trust anybody, and certainly more than yourself, probably, certainly more than me. So just avoid it.

But if you're a big company, you need to have some experts who really understand it. Obviously at Autodesk, the engineering teams-- every engineering team has security experts, and everybody in every team has to have quite a level of security awareness. Because it is so important. Half of the effort that goes into any web-based application is a security aspect. So when I say ignore it, I mean don't ignore it, but ensure that you can ignore it.

Yes. So this leads on to keeping it simple. That's really the message-- keep it simple. Simplify your data. Analyze the workflow and use existing components.

Here's another bit on simplicity. I just love this so much. This is my favorite topic. I love solving problems. And the only way to solve them is to simplify them. And if they're too complicated-- if I don't understand them, I won't solve them. So this is basically the main message also.

These are the three samples that I want to show. The first one is really old, and it's my first ever diving into cloud programming. So it's, source-code-wise, really, really bad. But that this is the one where I discovered how important the simplicity is, and how easily achievable, and how everything is available out there.

The second one is the simplest sample, the simplest possible sample based on an existing Revit SDK sample. Because all it does is it round trips one little number from Revit BIM to a cloud database via a web server. And in this cloud database, it's globally accessible. Because the cloud database is hosted by a cloud database hosting service.

And I can edit it on any mobile device in the browser. And with my Revit add-in, I can subscribe to that update and update the BIM in real time, if I so choose, which I probably wouldn't. That's just for demonstration purposes. It's possible. And yeah, with very little effort. And finally, the third sample is this Forge-based one I said, the 25 lines of code on top of Phillip [INAUDIBLE] boilerplate stuff.

So let's dive into. those. But first, one little caveat-- I'm talking about modifying some data in a BIM on the web and then getting that information back into Revit. I'm not talking about actually doing any BIM modeling work in the web. That's way beyond me and probably all of us.

So it's just about minimal bits of data. But if you are interested in actually building any kind of

BIM in the web, we're interested in that as well. And Forge-- I said before, Forge is read only. That's not completely true. There is one library that I'll mention again later on briefly called Design Automation. And it currently supports creation and manipulation of DWG files on the cloud through the web services.

And we are thinking about doing that for our VT files as well. So if you're interested in doing so, we're interested in talking with you about your needs. And there are some blog posts telling you how and where and what that is all about.

And Forge does have one little snippet of special RVT support already built in today. We can even see it live, which is that if I look at this-- here's a typical RVT file-- or not typical. It's a very simple one.

I don't know what this-- ah, yes. I'll click on this. This will-- if I double click it, it brings this manifest data that is my first entry point into extracting data from this model. I can also see the structure of the model here and dive into properties and stuff.

And if I right click it-- no. Here is Exports. If you look at these available exports, you can see this is an RVT file, and I can export to DWG. As well as OBJ, SVF is the viewing format for the Forge viewer. There's a thumbnail, and there's IFC.

So I can translate RVT to DWG or IFC with no Revit. That's built into Forge. It's just a week or two old, so that's one of the pretty newer additions. But that will grow. We'll be adding micro web services to Forge. Why doesn't this go forward? OK.

So that's enough about Forge for the moment. Now we'll go back three years in time to my first web-based connecting application. Who has seen this before? Quite a few. I hope I'll not bore you. I'm sorry about that. So I've been showing this for ages. And I was really fascinated when I discovered these possibilities.

And maybe before I show it, I can go on and say what was the biggest surprise, I said, when I started exploring this was NoSQL. Because this was in 2013. I was told by my bosses-- hey, do something with the web. You have to.

And I didn't, because people kept asking Revit questions all the time. But then I submitted a proposal for an internal Autodesk technical summit meeting, and it was accepted. And I was under real pressure to actually really do something.

So I googled for cloud database, the very first step. I was sitting in a meeting with my colleagues, and they were not aware. They did not know this. And I googled for cloud database, and up came NoSQL.

And I said, what's this? Weird. And this is pretty old now. And Twitter, Amazon, Facebook--none of them would exist without NoSQL databases. This is utterly fundamental. And if you are not aware of it but do any work whatsoever with databases, you have been sleeping really, really deep.

So the fact is that a large, distributed system cannot be built on transactions. And SQL databases are transactional relational databases. And they will not scale. That's mathematically proven. There's no way. You can build as fast computers as you want. It still won't help.

But there is something that will help, which is NoSQL. And this is not anything fixed terminology. This is just any kind of way to get away from transactions. So there are some typical characteristics. They're normally non-relational. They're normally distributed. They are, funnily enough, normally open source, scalable, and can handle huge data. And I might add, they're also very easy to work with. So it's really fun and cool.

The mathematical theorem that I mentioned is the CAP theorem. And the old database paradigm is called ACID-- Atomicity, Consistency, Isolation, and Durability. That is the transactional model. And the alternative model-- they've just been playing with acronyms here-- is BASE, so Basic Available, Soft-state, and Eventual Consistency, which means you might do things in a NoSQL database which temporarily might seem inconsistent, and nobody knows what the real consistency will end up being. But there will be a consistency. You cannot really mess up the system completely. But Amazon might sell two books at the same time. But they'll sort it out.

So now we'll use a NoSQL database. The first one I happened to choose back then is called Couch DB, Apache Couch DB. And the funny thing about that one is Apache already says it all. It's a web server. So it's really a database, but it's a web server. And you talk with this database through this web server interface. So you have it all in one package, which made it hard for me to understand, which is why I later switched to NodeJS and MongoDB where the web server and the database are two separate things. But in principle, CouchDB and MongoDB are quite similar. We'll see a bit more about that later.

So my idea for this technical summit presentation was I have a building model. I want to put some little piece of information into a cloud repository, and I want to render it on a mobile device, any mobile device. I'll use a browser. That's the easiest thing. That's available everywhere.

And I want to do more. I want to be able to edit it on the mobile device, update the cloud repository, and bring these changes back into the bin. And it would be cool if I could actually do that in real time. So I can do something on the mobile device somewhere in the world, and if I choose so, then the BIM is updated immediately with no other interaction. And that's easy to achieve.

A little bit more about the CouchDB implementation, just so you have an idea of what this kind of structure in such a database might look like. In these databases, everything is a document. And all the documents consist of JSON. So it's basically strings with a couple of braces and stuff.

Every document has a built-in ID and revision, so it's impossible to mess things up all too much. And the funny thing is the database design-- so the structure of the database and also the representations of the database-- I said this is a web server. So you have a view, or several views, into the database. All of these views are also documents, which means that the database is stored in itself.

It's one single, nice package. And the nice thing about that is it makes it really easy to replicate. You just take this entire package, put it into another CouchDB instance, and everything is there. And it's exactly the same, and there's nothing missing, and nothing to be added, and no configuration, and no nothing. So it's really, really easy to replicate.

And this design defines views and attachments. They're not so important. The views arr the important thing. And one important thing about the views-- I don't know whether I have a slide on that-- is that they use something called MapReduce, which are little snippets of code, little functions. They can be in almost any programming language. Often, they're in JavaScript . And they calculate-- they ask each document, do you have this property, do you have that property, and do something with it-- filter based on that or produce a result based on that.

And they are so small, and they are implemented in such a way that they cannot modify anything. And because of that, you can pre-calculate all of the results for all of these functions for all of your documents. Even if you have 10 million or 100 million documents, you know the answer to every query in advance. And that's the underlying mechanism that enables the speed and scalability.

So how did I choose? What did I choose to represent? I can show you on the screen in the browser. Basically, I have a model like this in Revit with a room like this. And I have a furniture instance, like this.

And those-- I wish to be able to edit the location and rotation of the furniture in the room, but on a mobile device. So my representation on the mobile device looks like this. So you can see I've extracted a little bit of graphics. I've also extracted-- I can actually click on that, and you can see it knows that this is a family instance. I can look at its properties. I can actually edit properties, and they'll update in the bin as well.

But the main thing is, first of all, I want to explain about what is that visual graphical representation. Because graphics is always cool and sexy. So to do that, I have the following BIM elements. And I reduce my information to only the absolutely essential information to represent that.

So I'm assuming I want to have 1,000 users with 1,000 models each with different levels and rooms and so on. So I'll have millions of records. So I have to be really efficient and structured about how do I filter for this stuff. So I create or look at these following objects in the BIM.

There's a model. There's a level which has rooms on it. There are rooms which have the family instances in them. Each instance is furniture or equipment, and it belongs to a room, and it also has a reference to a family symbol which defines its geometry.

And what I want to edit on the mobile device is I want to simply be able to move that family instance and rotate it. You can see that my rotation implementation is really crude. But that's all I want to do. And get that information back into the BIM.

So for the family instance, I have the placement or the location and the rotation. And that's all the information it has, because the geometry is defined by the family symbol. Plus, all of these might be equipped with some properties.

So where do I get those graphics from? From the room, I can analyze the boundary loops, including holes. For the family symbol, I generate one single bounding loop, the outside bounding loop. In case you're interested in Revit technical details, you'll notice one funny thing here, which is that this desk here, in Revit, you can see that it's just a rectangle, whereas in my web view, it has these little bumpy bits coming out, which are the handles of the drawer.

And the reason it has those is that I, for fun, implemented a really complicated way to get its boundary loops in that I didn't use the 2D representation. I used the 3D representation and project it onto the floor. And in 3D, this desk has handles, and in 2D, it doesn't. So that makes my add-in a bit more complicated when we look at the code, but it was a useful experiment. So that's the boundary loop that I extract from the family symbol.

And then I said translation and rotation on the instance. That's the data that I'm interested in. I also need to keep track of the relationships. So one handy thing about the CouchDB database is I can have these unique IDs. And if I wish, I can set them myself. And Revit provides me with unique IDs. So I can use those in the database. So I have a one-to-one connection between database and Revit, which is very, very handy, obviously, very comfortable and easy.

The family instance has to know which room it belongs to. The room knows which level and the level which model. This is very similar to Revit, also. There are some objects that have hosts, for instance.

And if you ask a wall, give me all your doors, it will say I don't know what doors are. I have no doors. You have to go to the door and say, hey, door, who's your host. And then you get the wall.

And this is exactly the same structure. So I was pretty much following Revit thinking here. And in the end, when I navigate through this system in the browser, I do need to go to the home, which lists all the models, and then go to the model. And in the model, I see the levels, and in the level, I see the rooms. And then the rooms I can pick before I get the graphics.

So I have to do exactly what I just described in Revit-- invert the relationship. So the relationship of the family instance knows that it belongs to this room, and so on and so forth. When I navigate down to it, I go to the room, and say who, what, where are your family instances, and it says I don't know. So I have to implement a database query, a view to give me that information. So that's how I navigate through this hierarchy.

And the family instance has references to two things-- the room that it lives in and the symbol that defines its geometry. So I implement that with database classes representing those things and with unique IDs which are exactly identical with the Revit unique IDs. So all of that is very simple.

One little additional thing is how do I represent-- how do I get this display stuff to show up in the browser. And I do that using SVG. That's Scalable Vector Graphics, a very sympathetic, simple format. And in this case, just to understand how it works, SVG implements one element called path. And the path has a definition.

And the definition is a bit like turtle graphics. So here is the definition of the path. And it says move to a certain point, line to another point, line to another point, and close. So it says move to 100, 100. That's probably this point. And then line to 300, 100. That's-- no. 100, 100 is probably there. 300, 100, 200, 300, and close. So it's simply a polygon.

So in my Revit add-in, for each of these geometric bits that I have-- the room and the family symbol, that's all-- I generate a string like that. And that's what I put into the database. So I can have a really efficient, super minimalistic representation of my graphical data.

So here's an example of such a data. This is a family symbol. So this is-- ah, yes. This includes some geometry. As I said, it has the unique ID. This is exactly the same as the Revit unique ID. It has a revision. The database generates that by itself and updates that whenever I make a change to it which I never will. Because the only things that I change in my implementation are the positions of the family instances.

And it has what I call a loop. And that's this SVG path definition. So as you can see, after the first line to command, I can omit line 2. So I have moved to xy, line to xy, and then I just go and say xy, xy, xy, xy, xy, xy, close. So very minimalistic, cool representation.

And that adds some information, some properties there. On the family instance, I have more properties, I think. I don't know why I don't see them here. But you see in the-- if I click on that and click on Properties here, you can see that I took all of the properties from the standard property set, actually.

Then there's this Views that I mentioned. Map and reduce-- I only use map. And then I define the views. As I, said I have to have some kind of method when I click on the model to find all the levels it owns, when I click on that level to find all of the rooms, when I click on a room to find all its furniture instances and display that information. So that's what I do with those views.

And all of this is implemented in JavaScript and populates this HTML scaffolding. So that is the entire HTML, I think, that I use to generate this web page. So now let's look at it live. There's not much to see, really, now that I've said it all.

So I have an add-in called Room Editor. And I have rooms. I have these commands-- upload rooms or upload all rooms in case I don't want to select them. Upload, update the furniture-- that means update from the cloud database after something has been modified. Or subscribe to the real time changes.

So let's simply upload all rooms. It picks these three here, and it generates that SVG information that gets stored in the database. So let's look at that first. I display that in these little .NET forms just so I know what I exactly exported and what it looks like.

And I also set a so-called sequence number. This is what I use. This is generated by the database. The database tells me, ah ha, you added something. Here is your current sequence number. And then I use that as a marker. Because later on, if I want to query the database for changes, I can tell it give me everything that changed after that point in time.

So now the database is up to date. Oops. And if I say Refresh, it will put that into its current position. I can modify that and say Save. That saves it to the database. And if I go in here and say Refresh, then it jumps to the new position.

So that's sort of interesting, not really hard. And the cool thing is I can subscribe to changes. And in that case, if I say Save here, then it jumps over by itself. So that's it. And all the code is on GitHub, so you can check it out.

But I said this was my first exploration into that whole realm of connecting data, desktop and cloud, and into web programming. So that code is not really, really nice, and I wouldn't really recommend that you use that exact approach.

Last year, when Forge started coming up, I dived into this again and thought, OK, I have to do it properly for Forge. And all my colleagues were using Node.JS and MongoDB for web servers and web-hosted databases. And that is a cleaner separation. You have the two separate, and I understand a little bit more about how that works. So I wanted to re-implement this for Forge based on Node.JS and MongoDB. And then when I started diving into that, I thought, OK, it would be handy to understand a little bit more about these two components first before I dive into the Forge side of things.

So I took a really, really simple existing Revit SDK sample called Fire Rating. Who is aware of

Fire Rating? I'm a little bit surprised, because it's extremely old, and it's the basic example of how to connect the Revit BIM to the outside world, but pre-everything, dinosaur, Excel single projects.

So the existing Fire Rating sample implements the first three steps there. It creates a shared parameter and attaches it to all doors. And the parameter is a double, so just one single number. And its called Fire Rating. So the idea is obvious.

It exports the fire rating values for all doors. That's the second commander. There are three different separate external commands. The first one creates the shared parameter. The second one exports the values to an Excel spreadsheet using the element IDs to identify the elements and adding that Fire Rating value and one or two other properties.

And then you can edit this Excel spreadsheet with the existing values. And the third command in Revit imports the modified values back into the BIM and updates the BIM accordingly. So very, very simple, very minimalistic. Just one piece of data for each door, and that's it.

And my idea was, well, this is a good example to take to the cloud. Because if I take it to the cloud and put this data into a NoSQL database, I can have the Fire Ratings for all my projects in one single place and do interesting analysis or whatever. And this is just an example. I might want to-- there's so many things that you can usefully deduce from having access to some aspects of your BIM but all of your projects in one single package. So that's why it's good to have NoSQL databases and scalability.

And obviously, if I do that, I cannot use the element ID to identify the doors. I use the Revit unique IDs again. And MongoDB supports me specifying the database ID, just like CouchDB does. So I can use the Revit unique ID here again in the database.

I also have to identify what project each door comes from. Because if I want to connect back, it's handy to know where do I have to put it. So let's say I put the data into a cloud database, I use the Revit unique ID to identify it, the cloud database is a MongoDB database, and I use Node.js for the web server. So now they're two separate components for the web connection and talking with the database.

And here again, just like before, I add another command to subscribe to the changes. In this case, there wasn't-- MongoDB did not provide such a handy marker number as CouchDB did. So I added another field, which is a time stamp.

Every time I export the information from the BIM to the database, I update a time stamp on each record so I know on each record, on each door, when was this modified. And then in my Subscribe to Changes, I can say hey, database, give me the values for all doors, but only from this project, and only updated after this time stamp. And then I'd get a pretty small set.

Here again, I had lots of interesting experiences. In the beginning, I wanted to be able to export the door data floor model and then possibly re-export it again to the database and leave the modified values-- leave the existing documents where they were. Just update.

And for that, I needed an export one by one. But each of these is an individual HTTP call. And then somebody tried this out in a realistic project with a couple of thousand doors and said, hey, this is taking a long time. A couple of thousand HTTP calls is not a really good idea.

And I researched a bit in the database, and I could not find a way to send all of them in one go without deleting the existing ones. So I was reluctant to do that. But in the end, I did. So that's why I have two different export mechanisms here-- export one by one, which is fine, if you have a handful, but if you have more than 20 or more than 1,000, you definitely want to export by batch.

And then it's two calls, two HTTP calls. I don't remember why, but there are two of them. And one of them is actually package all of the data. And all the door data up for 10,000 doors for a serious project is not much. If you squeeze it into JSON or whatever and send it over HTTP, it's a very small, small package. So there's nothing to worry about in that scenario.

This is an overview of the architecture that I talked about. So I have Revit with its BIM. I have the Revit add-in. I isolated part of the connection to the web server. I isolate it into a separate little class. Because once I had that set up-- I was in Madrid last January, I think.

And another guy came and took my code and implemented a standalone Windows client which accesses the same database. And it was so easy. It was like, again, 20, 30 lines of code to really have full control over all of the information in that database.

And when he did that, I said, well, we can't have the code duplicated in those two add-ins and potentially in hundreds of others. So I'll isolate the little access module into a separate class library and use that to connect to the web server. And then the web server translates it onto the MongoDB database.

And the handy thing here-- I mentioned that briefly-- is that the MongoDB database can be hosted by a free database hosting service, which includes an editor. So I can show you live how you can access this information.

What else do I want to say? Let's go over to Visual Studio and actually open that project, just to show you how small this code is again. So I'll go to a different project, which is stop debugging. And Open Recent Project. And this Fire Rating cloud.

So this Fire Rating Cloud solution includes three separate projects. There's the Fire Rating Cloud Revit add-in. This is what you typically notice. It has an external application. It has the commands that we saw on the screen before, and so on and so forth. Then there's the Fire Rating client, which is this Windows form thing that I mentioned.

And then there's the Fire Rating class library, which has a utility class which does the web connection, and a door data class which defines exactly what information do I want to send across from the desktop to the cloud. And this is the entire implementation of that.

So you can see it's six, seven, eight lines of code. That's the main data repository definition. Let's go to the web server and look at that. So here we have the web server, for instance. But maybe first, we'll look at the model.

So the database definition-- this is a JavaScript module with 594 bytes. So you can imagine the tremendous complexity of this. And it looks like that. So this defines my database structure.

And you might notice the similarity-- ID, project ID, level, and so on, ID project, ID level. So what else do we have in our web server? We have the server itself, so Server. Require a couple of libraries-- all of this is open source.

I can switch between local or a web-hosted database. If it's local, then I use MongoDB directly, local host. The Fire Rating database-- if I'm on the web, I have a different URL. This Boolean is the only thing I set to switch between local and global database.

So I can do everything that-- every single one of these samples that I'm showing, I can run locally. I can isolate it into my own intranet. I do not have to put any information on to the web. Actually, I should have mentioned that before when I talked about security. If you worry about security, then disconnect and stop worrying. All of this stuff is available locally. You can run everything on your own servers.

And I connect to something called Mongoose. That's just a library that wraps the database. So this is basically the database. And I need some stuff to connect with the web, and so on, and to handle requests coming through the web. But this is the entire server.

So here at the end, most of it is actually creating this message. And I say require the model door. That's the database definition that I talked about. And then there's the roots. So the roots define the HTTP REST access points that I implement. So let's look at the roots.

This is the entire definition of my REST API. So I have these different URLs. These define end points. I call these URLs, and they do things for me, because I can supply them with information. And each of these URLs-- so for instance, list all the doors, application GET, API version 1 doors is the URL. There's a version in here. Because if you implement a REST API, you don't want to change it. You could have thousands of people out there in the web using your API. So you can't change it, ever.

So if you change it, you don't. You create a new one and call it version 2. If you look at the Forge URLs, you'll see the same thing. They're all version 1 or version 2. And if they ever get changed, then they'll be-- they won't change. They'll be version 3.

So this you URL gets rerouted to a function in the door service which is defined by controller doors, version 1. So let's look at that. This shows you how the REST API endpoint is connected to the database wrapper. So to find all doors, the first one we looked at-- I have a request, I have a result, and I say door find. And I have a filter which specifies nothing, which means this will return all my doors in the database. And that's it.

Basically, I just want to show it is very, very simple. So let's run this. I can-- well, yeah. Let's go into there. I was talking also about the different modules here. So there's the Revit add-in, the Fire Rating library that connects to the web. There's the utility function that actually does the URL stuff. Here again, I have an option to switch to a local web server. If you look at the architecture, I have two options here, two little bits. I wish I had a pointer.

There's the web server. I have a switch to say is that local or global. And there is the database. I have another switch to say whether that's local or global. So I can put both of them on my local machine or just the web server on my local machine, but the database on the internet, or both on the internet. And that's two Boolean variables and nothing else.

So that's why I have that switch there. And I have the API version defined. So I use that to create my base URL. And then I have the different queries that I make. So I have a put and a get. And this could be made even shorter and more elegant in the delete and the Post Batch. But they all go through the same thing, basically.

And then there's this other interesting little aspect, the Windows client, which uses the same library to access the same database and simply displays it in a web form. So let's go to the Revit add-in first. And I'll say start in the debugger.

And this will load some model-- I think it's a very simple one-- with a little door in it. And we'll store that in the database, and then we can take a look at the resulting data in the MongoDB database hosted on the web by such a web service. So there's my door. Where's my add-in? Room Editor, Fire Rating Cloud.

So as I said, before I do anything else, I have to create the shared parameter. If we look at this door to begin with, you'll see that it-- ah, it already has a fire rating, so this has already been defined. OK. In that case, I don't have to create it. I can export all doors in here.

Will it tell me a message? Null reference. I guess I should have run this before. Ah, actually, I think I do have to create-- whoops-- set up the Fire Rating. I think there's something messed up here. Bind-- so now I have two Fire Rating values.

Oops. This will get a little bit messy. Sorry. I should have started it in a new project. Maybe I should create a new project for safety's sake, or maybe not.

And so now I created a new Fire Rating value, which has a value of zero right now. I'll export all of these. The non-reference was because it was searching for the parameter, the shared parameter definition, and it wasn't finding it. So I don't know why something can be in there that it's not finding, but that's the case.

So now this went through the-- this Fire Rating web service to the cloud somewhere. It's running, I hope, because I have it hosted in Heroku. I haven't looked at it for weeks, but I assume it's out there.

And that Heroku thing is passing things on to Mongo Lab or MLab. So I log in there and hope that it takes me where I want to go. So I have two databases running here, and they seem to be alive. And in this one, I just have one single door now.

And here you can see the fire rating tag is currently set to zero. So that looks fine, and you can see this time stamp here. That's a Unix time stamp, the one that I mentioned to keep track of changes. And now I need to remember-- ah, yes. I think I can just edit this and say OK, the fire rating value is now 120. Save this.

Let's actually make that smaller and go into Revit and say Import. And you can see the 120 shows up there. So we seem to be live and connected and well.

I'll move that over to there so we can do more interesting stuff, as in, say, subscribe to changes. Time stamp set to this value-- so this is 1976 at the end. Now it gets complicated, because now I want to say, OK, 240, 1976. So let's say 2000. We don't care about real things.

Veracity-- where do we save? I need more buttons. So this should update right away. I possibly may need to go across with this cursor. I may need to click. Ah, no. I haven't clicked yet, but it updated. So you can see I didn't write this. This is a public Mongo lab hosting service.

But I can use it to connect my data from anywhere in the world to any BIM that is running my add-in. You can see that I did very, very little work, actually. The programming-- the lines of code are less than-- well, a few hundred maybe, 100 lines of code, but not much. Yes, please.

AUDIENCE: So when you updated the database, does that make it change to your server, and then that server [INAUDIBLE] change?

JEREMY TAMMIK: I have to think.

AUDIENCE: How did the database chain cascade through the server if you're listening to changes in the database?

JEREMY TAMMIK: That's a good question. I have forgotten, to tell the truth. Let's look at that thing there. Maybe that'll help me think.

It must be something like that. I'll have to look in the source code to answer your question. That is a very, very valid question which I have also wondered about. I've pondered that in the implementations. I insert, I update-- update, send status, delete, find all for project-- the server-- the server isn't doing anything.

I really forget. I really forget. It's in there somewhere. There's not many lines of code to look for. But that is absolutely the point of the whole thing. I really wonder what I did there. But that's what that time stamp is for, and that's why I had to update. If I hadn't-- I think, actually-- maybe if we look at the Revit add-in-- oh, yes. Yes, yes. I can tell you-- yes, I'm pulling. In this case, I'm pulling. And that is not really nice. There would be a better way to do it.

I have the BIM updater here. And if I look at the subscribe, I say total subscription-- yes, yes, yes. I have-- I'm basically using the idling event, but I recommend to people do not use the idling event for this kind of thing. If you want to pull, use an external event instead, and run it through a Windows timer in a separate thread, because then you have control over the frequency.

If you use idling, it's fine if you do once only. But you do not want to pull using idling, because it basically goes into an infinite loop and brings down performance, hogs the CPU. So I use an external event. Very good question. Thank you for that. So long ago since I did all this.

And and there would be possibilities to implement push notifications from the database. And that is what you should do. And that is what I-- in the Forge sample, which I'll come to next, I use Socket.IO to broadcast changes. That's one way of avoiding to use the push notification from the database. But the push notification would be equally fine.

AUDIENCE: That status was set to Subscribe within the project, and that [INAUDIBLE] has closed down, and Revit itself it closed down. Would that also update?

JEREMY TAMMIK: No, absolutely not. No, no. This subscription thing is running inside of Revit. So I have an external event which gets triggered every couple of seconds or a couple of times a minute. And each time it goes and pulls and updates the BIM.

But what I would have to do if Revit is closed down and no add-in is running, then I have to have some kind of marker in the database and in the BIM saying the BIM is this status, the database is at such a status, synchronize. But I can implement that.

AUDIENCE: If I set that subscription, is it set at the application level, or is it set at the project level? So the next time I open a Revit project-- let's say I open Revit project 2, or yesterday, I set the status to Subscribe. Would it update [INAUDIBLE]?

JEREMY TAMMIK: This is no setting. This is just code running. And you'd have to run the code in the project the way it is right now. But that's really-- you have all those choices yourself. If you implement this, any kind of system like this, you can choose whatever you like, and you have all those options.

Good. How are we doing for time? Let's see. OK. So we have 25 minutes left. Good, good.

OK. Let's look where we are. Ah, yes-- it might be cool, but you can understand the concept, I guess. I'll quickly dive into that as well. So let's start up the fire rate, the standalone client also so you can see-- does the rate say run or something? I won't say Run. I can go to the command line, I guess, and do it there.

Fire Rating Cloud. What do we have in there? The Fire Rating client. Bin, debug, Fire Rating client. So this is the standalone thing which I can also use to edit in the same way. This is talking with the same database, going through the same Fire Rating HTTP connection library and displaying the same data.

So this client is written in-- I don't know what-- 20 lines of code or something. Let's actually take a look. So this is the main line of the program. It does nothing whatsoever. And in the form-- this shouldn't do that. That should say view code.

You can see here this is the entire implementation of this standalone client, which it gives me a possibility to edit the database from a Windows form. But I could implement anything I want. I can put this editor onto a mobile device into a browser, into JavaScript, anywhere. It'll never be more than 20, 30, 40 lines of code.

So I have global access to one piece of data stored in a globally accessible database, and I can hook it up with a BIM. I can update in real time. I can connect the desktop with the cloud. And it's really, really simple.

So should I stop saying that? Next step is Forge, a platform. Does anybody not know what Forge is? Have you managed? Was it possible? Not quite.

So Autodesk started implementing this a little bit more than a year ago-- or publishing it maybe a little bit more than a year ago. We had this DevCon, the Developer Conference in June in San Francisco. That was pretty cool. And that was really the point at which a lot of stuff stabilized.

So now we have a very clear idea of where this is headed, and we also have a lot of enthusiasm and a lot of success stories already with it. So it's really about providing a platform for developers to work on to access data globally. The advantage over the samples that I've showed is that in the Forge translation of a BIM model, you have the full data.

You have everything. You have all the graphics, all the objects, all the properties, all the relationships, all the model structure. Absolutely everything is in there. Whereas in my thing, you saw I had to pick out-- I want to export one little piece of information.

But this fits also in very well with the BIM idea that I showed before about the number of users. I'll just show you quickly this one slide, maybe before-- what are the advantages of a Forge-based application versus the do-it-yourself that I've presented so far?

So we have a realistic model rendering. We have 2D and 3D views. We can switch between the two. We can synchronize them with each other. We can display them side by side. And if something is picked in one, it can be highlighted in the other, and so on. So if you navigate in the 3D view, you can update the 2D view to synchronize it and everything. You saw all that, probably.

And that's it, basically. Aha-- you have OAuth so you can access other people's models. And I said my implementation here was a minimal amount of change on Phillip's boilerplate code. So the current forge APIs that are available are listed there on the left.

So there's authentication for providing credentials to access model, data management. All of these are REST APIs except for the viewer. The viewer is a client-side API in JavaScript for modifying what the viewer looks like. And all the others are REST-based APIs.

The viewer is completely open source. The others are partially not open source, or maybe not at all, basically. And also, they run on Autodesk servers. So there's no way to currently do anything with Forge without at least once passing your CAD model on to the Autodesk server and performing the translation.

The result of the translation can be stored locally, and then you can cut off. And then you never have to talk with Autodesk again if you want. But then you have to manage some stuff yourself, of course.

And what I use-- I mentioned that the design automation API currently supports creating and modifying DWG files, and that we're thinking about extending that for some kind of RVT or RFA access or creation. That's in the pipeline. It's still a ways away. I think we're talking about doing Inventer first. But things are moving really fast in all of this, so it's not that far away.

And the other three are the ones I'm interested in. I need to authenticate myself to access my models. I need the data management API to upload the model and translate it. And I need the model derivative API to extract the data out again and also get the stuff that I need to display it in the viewer.

So this is what the Forge-based BIM Editor looks like. As said, I'm using Phillip's-- why am I not-- oh, yes, I am where I want to be. I'm using Phillip's boilerplate code, which does all of the stuff that I've shown so far. And if I double click here, then it brings up my model in the viewer. And this is the standard Forge viewer. I can click an element, and I have its. properties.

I can explode things, I think, if I want. So I can see the model. I have the model browser. But Phillip also implemented a model browser here in this hierarchy thing. So that's the standard viewer and all of the CAD information. So if I go to one of these things here and look at the properties, you'll see it includes all of the information that is of interest to this wall.

So let's connect this to our BIM again. I said Forge is a read only system, so that model is uploaded to the Autodesk server, translated, and I get information back. And that's it-- one way only. And my addition is to implement a little round trip functionality. So I have to stop debugging and start up a different add-in, which is-- where are we? Recent Projects, Room Edit 3D application.

So this is a very small application again, very simple. Because here, I don't have to do any exporting, because that's handled by Forge. So the only thing I want to do now is update the model. And here I update it more intelligently, because I only listen to a socket I/O notification.

So I'll start up Revit. I'll set this as the startup project and launch Revit in this model. And in the Viewer here, I have a couple of additional buttons. So everything you've seen so far is just boilerplate code demonstrating how to program Forge in general.

But I enhanced it by adding a Room Edit 3D V3 Viewer extension to it. So in this viewer, this is all JavaScript. And if I hit the Debug JavaScript console, I can analyze my code, and see what's going on here, and who is programming what, and what is the values of the different variables.

And if I click the Start button, it loads another plug-in here, the Transform Tools. How can I get away that explode thing? So these Transform Tools enable me to pick something to translate it.

And now if I click, I'm not rotating the model anymore. I'm selecting something, and I can move it. So I said before Forge is read only. And also, the viewer is a viewer. You can only view things.

But the viewer supports extensions, and you can program these extensions. They're very small. And Phillip has implemented at least 50 samples already. And this is one of them, these Transform Tools.

And now if I go to the JavaScript console, you can see that there are some messages showing up here-- yeah, exactly four. And they're exactly the interesting ones that I want. And so one message is I loaded my extension. The next message is I've moved an element.

And this element translation has been logged. And the Viewer, which is on my client, on my local machine, communicated that to the web server. The client, that's running in the browser, but the whole application is running on Heroku in a web server. So there are two different pieces working here.

And this is from the viewer, and it communicated it up to the server here and broadcast it through something called Socket.IO. And to listen to Socket.IO broadcasts is very simple.

And I'll go to the debugger, , or to the Viewer, Command, and Command, Toggle Subscription-- OK. I guess-- ah ha. BIM Updater, Translating-- this is the actual updater-- NQ, Application. Ah, here it is, I guess.

So yes, I want to listen to a Socket.IO broadcast. So I have something called IO, which is a QU object Socket.IO client .NET. I have a library here. So once again, everything is already done for me. I don't have to do anything myself. I use the QU Object Socket.IO client .NET client.

And I tell it I am listening to a specific URL. And I listen-- when I receive a message-- or when I can connect, I say connected. And from then on, I listen to the transform message. And that transform message is this broadcast by my web server on that URL to the universe. So any client anywhere in the world can listen to it. And if my add-in is running in Revit, it listens. It hears that broadcast, and if I so desire, updates the BIM model in real time.

So let's see if we've got Revit up and running and which one was it. I should pick the right-- Room Edit 3D, I guess, is this one. I should pick the same model since it's communicating via this unique ID, so it needs to identify the elements.

So if I have this up and running, do I have to-- ah, yes. And I have to run my add-in. This one has one single command. So this command here toggles the socket IO subscription. So now I say subscribed, and that's it.

And now I need to get Windows in parallel with my Mac browser. What's that stuff down there at the bottom? That's-- I just want to see this model. I guess that's enough, really. And move this over there. OK. So now if I edit this wall and move it over there, I may need to-- ah, yes. OK. Good.

So you can see that it didn't much like that change, so I have to cancel it. You can see the BIM is still a BIM and still has its intelligence. Let's go into a top view instead, and here as well. How do I get to a top view? OK. That looks-- wow. This has never worked so well before.

Now if I click that and say move it over there, this is totally [INAUDIBLE]. I have implemented absolutely no-- why is that not updating? I've implemented no controls over anything. I'm just moving an element in an arbitrary Way but you can see the principle. I wonder why-- ah, I didn't acknowledge the warning.

OK. Now that other table moved away as well. I should move something that doesn't protest this. Oops. This should now maybe be happy. Hello? OK. I had to click. Sometimes, I don't have to click. It depends. But it doesn't really matter from the principle. So that's it.

Let's see. Eight minutes of questions. Do I have any more slides? I have a couple of more slides as well. I might take one or two minutes to go through those. So there's the evolution. Aha.

I have written a blog post which highlights exact lines of code that I changed from Phillip's boilerplate to my changes to implement the Socket.IO thing. So if you're interested specifically in doing something in a Forge model and communicating that information back to the BIM, there are those 25 lines of code that I mentioned easily accessible.

Here are the repositories. So you can-- they're all documented on the building coder. If you want to get started with Forge, these are the four entry points. developer.autodesk.com has everything. That's the main entry point. There's the developer conference. There's the Forge booth, and you can tweet. And if you have other questions, there's an answer bar.

And now I'm open for your questions. Yes, please.

AUDIENCE: This might be a simple question. Listener toggles [INAUDIBLE], right? Do you have to follow that every time you [INAUDIBLE] the model?

JEREMY TAMMIK: The way I've implemented it now, yes. But you could easily change that. That's totally trivial. You can automate that if you want.

AUDIENCE: [INAUDIBLE] bi-direcitonal, so somewhere [INAUDIBLE].

JEREMY TAMMIK: Ah, yeah, good question. Can I go bi-directional? And with the samples that I showed, the ones that I did myself, you can easily go bi-directional, totally trivial. With Forge, you have the translation step. And that's a little bit heavy, depending on how big the model is. So translating my model takes maybe 10 seconds if it runs well, or 30, or a minute. But if you have a real BIM, you could be sitting there for a couple of minutes or longer.

So you couldn't have a really good bi-directional, real time experience. But you could have bi-directional in the sense that the team keeps up to date and they have a lag of maybe five minutes or so. That's absolutely achievable. Question from the left side of the room, from my point of view. OK. Then it's you again.

AUDIENCE: What's the difference between MongoDB and CouchDB as far as your experience goes?

JEREMY TAMMIK: Yeah. Ah, that's an interesting question which is probably of very little interest to everybody else. And was there a significant difference between CouchDB and MongoDB? Not really, except I think CouchDB-- I looked at it three years ago, and I understood nothing-- is a web server with the database integrated.

It's probably highly, highly, highly super efficient. MongoDB is simpler, and runs in its own space, and has nothing to do with the web server. But as far as the document handling goes, I found them absolutely equivalent.

I had little details. Like in the MongoDB, I made use of this database marker, and I thought that was handy. I didn't have that in Mongo, so I had to implement my own time stamp. And that added some complexity for me. But it also means I have some-- I have more flexibility. I have total control. MongoDB seems maybe a little bit smaller, freer.

So you can really do anything you want. In CouchDB, Tiny a tiny little bit more structure is already in place. But I would say they're both great, absolutely great tools. And I don't know about others. I would just read about-- I don't know much about this. I'm not the right person to ask, either.

AUDIENCE: Just as far [INAUDIBLE], both of them.

JEREMY TAMMIK: Yeah. I love them both.

AUDIENCE: Piggyback on that-- do you know anything about Amazon's [INAUDIBLE] and how it compares to either of those?

JEREMY TAMMIK: Do I know Amazon's DynamoDB?

AUDIENCE: That's another [INAUDIBLE].

JEREMY TAMMIK: Yes. No, I've never heard of it. picked Node plus Mongo because it's totally mainstream, and all my colleagues are using it. But there are other really exciting ones. I just ran across developers working with some graph database.

It has about-- the name with about-- Neo? Yes, Neo for J. I was going to say it has a name with about four or five letters, but I can't remember which they are. That seems great as well, without having touched it at all.

OK. Well, I won't keep you if you have to go. If anyone wants to--

AUDIENCE: Without opening the RVT, can we update the RVT?

JEREMY TAMMIK: Without opening the RVT, can we update the RVT? Nope, you can't. But you will maybe, one day, when the Forge design automation API supports RVT. But currently, no. No way.

So thank you very much.

[APPLAUSE]