**w04-01-SystemDesign-2024**

0:00  
OK, so this morning we're looking at system design.

0:04  
The idea is, as I said last time, we're walking down one software iteration, effectively thinking about how we develop something from nothing at all.

0:14  
So we've got a basic idea into a piece of software.

0:18  
Now, later on in the module, we'll talk about iterations of the life cycle loop, but today we're continuing to walk down this.

0:26  
And so today we're looking at system design, which is one stage beyond a user interface design.

0:35  
OK, so the user interface design, what should it do?

0:39  
It should ideally be loosely coupled to the framework, that is the software libraries or the infrastructure that you're using to build it.

0:47  
So you should be able to design your user interface design.

0:50  
There's a low fidelity prototype.

0:51  
Not actually worry too much about the actual technology that you're going to use.

0:56  
And then afterwards think about the best technology.

0:59  
Now, another aspect of system description is that of the data model.

1:04  
Now the data model is a set of classes that hold the data.

1:10  
It could be data that are held in a database.

1:13  
It could be data that you're holding in memory, and you're going to work on those data.

1:19  
Now it's not all of your classes, It's not all of the data you have in your application, but it's the key ones which are being passed around your application that are holding values and you need to understand that.

1:32  
And it could be that they're also decoupled from the so-called serialisation choice.

1:36  
Serialisation is the process of storing data.

1:39  
So you have them in memory and then you record them to a database.

1:42  
So you might record them to a file or something else, but when you're coming up with the data model, you're just thinking about the data that you need to hold or store, irrespective what you actually do in terms of technology later.

1:57  
And then the next step is to think about the architecture, That is how the application is built on the big scale.

2:06  
I mean, if you imagine a building, somebody might draw a sketch in terms of the number of floors.

2:13  
You have a general idea.

2:16  
They don't tell you all of the details about how perhaps the window frames are going to be fit into the building, but there's a general idea of how you're going to build it.

2:24  
An analogy with an actual building would be you could choose an approach.

2:30  
So for example, you could choose a steel frame and then onto that steel frame you could attach something.

2:36  
Or you could choose to build with brick up from the ground and then on top put a roof.

2:43  
So this would be an architecture approach and this effects how you implement the software.

2:49  
So that's why you think about it to start with.

2:52  
You do perhaps think about it again and again, but it's very important that you think about it carefully at the beginning.

3:00  
And then the last thing we're going to talk about, not today in the future week, is describing the functionality.

3:08  
So bits of things that you want inside the architecture, they're a little bit more low level.

3:15  
You can also describe them without discussing actually the implementation.

3:20  
And it is more the sort of potentially low level design.

3:24  
So you have high level design, you have architecture as well, and then low level design, which might be thinking about functions and such.

3:33  
All right.

3:34  
So the data model, the application has a series of data models.

3:38  
Potentially these are associated with a user interface where you have input data or data you're displaying.

3:46  
They might be exchanged between services.

3:49  
So for example, if you have a web service, you send some data normally is Jason these days, but you send some data.

3:56  
So you have some classes that are associated with those data.

4:00  
They may be application programming interfaces.

4:03  
So that means that you are calling another function and you're sending the other functions some data.

4:09  
So that's part of your data model.

4:10  
And as I've already said, serialising that is storing the data somewhere.

4:15  
That's very important to understand what you're going to store because you might store it in the wrong way otherwise.

4:24  
So we need to document this data model before we go ahead and implement the code.

4:29  
One of the big mistakes in projects that have gone wrong is not understanding the complexity of the data model because it underpins things.

4:38  
So for example, the Police Scotland problem that you saw in the first lecture, that was a lot about not understanding the complexity of the work that was needed.

4:54  
OK, so the user interface data model, how do you get that together?

4:58  
How do you describe it?

5:00  
Normally you talk about how the user interface is going to be used with the people who are going to use it.

5:06  
So you need to think about the type of data.

5:09  
Are these data dates?

5:12  
Are they numbers?

5:13  
Are they floating point numbers?

5:15  
Can they be negative?

5:16  
Can they be positive?

5:17  
Can they be missing?

5:19  
Now in a database, you can actually store the concept of a missing value.

5:25  
This is called a null.

5:27  
In Python, it's non.

5:28  
But a lot of languages, modern languages that is, have the concept of a null or a non.

5:35  
So it could be that you're allowing this thing to be empty.

5:39  
For example, a middle name on a registration could be empty because not everybody has a middle name, whereas a surname definitely can't be empty.

5:48  
So you need to think, do I need a value or not?

5:53  
And if I do need a value, what should it be?

5:55  
And then you want to verify it.

5:57  
So this is a user interface feature.

5:59  
But out of that, you then have a potential data model.

6:03  
You also need to think about from the user interface point of view anyway.

6:07  
Is it an input, something the user's going to provide, or am I displaying it?

6:12  
Or sometimes you can have something that's both, so you're displaying it, but the user's allowed to change it and then save it again.

6:22  
So often the data model's associated with the states of the user interface.

6:26  
So you could be in administrator mode or have some kind of special role where you're allowed to do something, which could be part of the data model.

6:35  
Now, normally if we're dealing with databases, that is a relational database.

6:40  
There are other types of database that are not relational.

6:44  
But if we're dealing with a relational database, we often have relational entities, meaning we can describe a set of tables.

6:52  
The word relation is normally implemented as a table.

6:59  
So you have a set of tables of data and they relate to each other.

7:02  
However, if you're not using a relational database, let's say you are thinking about using a no SQL database such as Mongo DB or something else that stores data in a different way, then you might not want to worry about an entity diagram.

7:17  
You might instead want to use a class diagram.

7:21  
So this is something you've already seen.

7:23  
In fact, in nine O 5, we saw a little bit of that class diagrams with UML, where you describe the key classes with UML, what you want to store in them.

7:33  
Now, at the point of using these class diagrams, it's not actually required to use any particular language.

7:41  
You could just draw them and then afterwards think, Oh yeah, that needs to be Java or it needs to be Python or some other programming language like C++ doesn't really matter.

7:53  
Now we do need to talk about potentially the data structures that are underpinning the application with the client.

8:01  
Now this is only normally useful when you have some client who really understands the processes, but it's still a good idea to echo back to them what you think are the data that are being stored and what you think are the relationships between them.

8:17  
I had this happen myself with a client and I did try but the client was not being very clear with me and that really made it rather hard to build the software.

8:29  
So understanding this is a good thing.

8:32  
There is actually, I'd say a way of doing this with interviews where you sit the client down and you do it formally a little bit like a user interface workshop.

8:43  
But you really do need to understand the key relationships like can you have 1 of this and many of that?

8:49  
Or is it always a one to one?

8:51  
Because that will be built into the the bedrock of your application.

8:57  
All right, to remind you, here is a UML class diagram.

9:01  
And in this case we have a person and the person lives at 1 address and the address can have zero or more people.

9:12  
I'm sorry, zero or one person living there.

9:14  
So it's a bit of a odd UML diagram, this one, but it's just to show you you can have these relationships.

9:20  
Now, these two down here, they are inheriting from the person.

9:25  
So a professor has all of these attributes and also a salary, whereas a student has all of these attributes and some other details.

9:35  
So that's just a reminder.

9:36  
You can always go back to the textbook if you want a little bit more.

9:42  
This is the entity diagram which I tend to like drawing.

9:45  
There are different ways of drawing entity diagrams.

9:49  
Some are more or less clear to the client.

9:53  
I tend to use this one when working commercially just because it's easier for the client.

9:58  
It is drawn here with so-called crow's foot notation.

10:02  
So it's a crow's foot because if you think about the bird, the crow, it's got several toes and these are the crow's feet.

10:11  
So the idea here is that the customer can have many orders, but the order belongs to 1 customer.

10:20  
Now the syntax here can mean zero or more, or in this case it's exactly 1.

10:27  
But this could be this is actually zero or more.

10:30  
All right, So the little symbol at the end does mean something.

10:33  
You can also put, if somebody doesn't understand Crow's foot notation, you could put the numbers next to it anyway, so the customer has zero more orders.

10:44  
The orders comprise zero more shipments.

10:48  
That's the idea.

10:49  
And so here we have a set of relations or tables.

10:52  
So in this case the relation database would have 3 tables.

10:57  
The FK is a foreign key, so that means it's defined in the other table.

11:04  
So PK is a primary key, so the value here is put in there and so that's used to form the link.

11:12  
So this foreign key has to be in the other table anyway.

11:19  
So once you think you understand the data model, you then have to have a few goes at it because often clients aren't very good at spilling the beans.

11:29  
You know, they're not very good at telling you all of their thoughts about the data model in one go.

11:34  
So you may require several consultations.

11:37  
And yes, you probably want to find out who are the technical stakeholders, like who are the gurus, the people who really understand the data model.

11:46  
Don't bother with the product owner or whoever.

11:48  
It is high level stuff.

11:50  
You want to go for somebody who's really understands the process.

11:54  
It might not be somebody who understands computing very well, but it might be somebody who really understands all the data relations.

12:02  
So it could be an admin person who knows the data very well.

12:06  
And as I've already said, you need to verify those relationships.

12:11  
So then we can create a data model with, say, the input data that's going into our application.

12:17  
And we can do this without the software.

12:19  
We don't have to worry about the technology we're choosing.

12:22  
We could share this with the client using, say, Excel.

12:27  
A lot of people still use Excel.

12:29  
So you could put your relationships in as different tabs or different sheets in a worksheet and you can put some fake data in there, you know, guess a few values, give it back to the client and say, is this right?

12:42  
Is that the relationship that you have inside your data?

12:46  
If you are dealing with something that's not relational, meaning you end up with some document format, you could give them Jason or some other class diagram way of describing the relationships.

12:59  
Now if you are dealing with relational data, you do want to check the normalisation.

13:05  
Now they may have not mentioned normalisation yet or maybe they have.

13:09  
The idea here is we try to try to avoid repeating ourselves in a table basically.

13:17  
So if you have a one to many relationship you keep the things that exist one in this table which is 1 and the things that are many in the many table.

13:28  
So you have to look at your tables and decide am I repeating myself?

13:33  
Have I only got 1 foreign key here or have I got actually several and I only really want one?

13:38  
Anyway, I'll leave it to the database lecture to discuss this more.

13:43  
Now you need to fix, often fix the price of building the piece of software and that is often connected with the complexity.

13:52  
So as I've already said, if the data model complexity is wrong, the guess about the cost of the software will be wrong.

14:00  
And finding this out later is very painful, financially painful obviously, and relationship painful.

14:09  
So the data model implementation we want to hold what's transient potentially these are data we've read from somewhere, They could be read from the database.

14:18  
We've calculated something, and then we're going to give it back to the user interface.

14:22  
So often you have data that are in the user interface, you have data that you're storing, and then you might have some data in the middle that you've calculated because you've added some numbers together and multiplied.

14:32  
So these are referred to as transient because they're not actually stored and the user doesn't actually see them either.

14:39  
But they are important.

14:41  
Imagine you are storing map coordinates and you read them back into memory, and you want to calculate an angle between two map coordinates.

14:52  
You don't need to store that in the database because you're storing both pairs of map coordinates.

14:57  
But you do want to calculate it because you might then want to display something into the user in terms of a compass bearing.

15:06  
So we want to try to minimise the complexity of the data model as much as we can.

15:11  
We don't want to be copying data all around the place.

15:14  
We want to think about the data flow carefully.

15:16  
Where are the data going in our application?

15:20  
And ideally we use automatic generated documentation from the implemented data model.

15:26  
Now why do we do this?

15:27  
Because continuously drawing UML diagrams for more and more complicated data models is a waste of time.

15:35  
It's just going to be a time sync.

15:37  
And so if you implement classes with comments in the appropriate form, we're going to come back to this later on.

15:45  
In the module, we can generate what's called a documentation string, and from that we can actually generate our documentation, and then we can generate the UML diagram from the code.

15:59  
When you're first building an application, you probably want to design the UML diagram yourself manually and then discuss it with the client.

16:08  
It won't be all of the data classes, but it will give you a good start.

16:12  
But then once you start implementing it, you probably want to just generate it from the code.

16:16  
Now, I've had jobs where I've been called in to fix software that somebody else has written, and there was no documentation.

16:24  
And so one of my first starting points was to automatically generate all of the UML diagrams out of the code and to give myself a bit more of an understanding of what was going on.

16:38  
Yes, I can read everything, but it's a lot easier to see a sort of map of what is happening in your software as you're reading all the other files.

16:48  
OK, so that is a brief discussion of data model design.

16:53  
Does anybody have any questions or comments about it before we go on to architecture?

17:02  
Yeah, Yeah, good question.

17:20  
So when it comes to a data model, this sort of fit initially fits under requirements capture.

17:26  
So you might have a specialist involved in relational databases who's good at this sort of thing.

17:35  
Often you need people who are good at talking to clients, you know, good at the people person.

17:40  
So you remember the guest lecture last week?

17:42  
We had two people.

17:43  
There was the Scrum Master and there's a product owner and yeah, you might need something like that.

17:48  
So I've conducted workshops with clients where there were two of us and sometimes you need it because the one person has got the technical and the other one is trying to talk to the person.

18:01  
Occasionally have it in the one person where they have both abilities and they can be just fine.

18:06  
Often whenever you conduct those interview sessions, you take notes and then you go away and you create a new model and then you go back again because it's rather hard to take.

18:19  
If you're going from very little, it's rather hard to take in all the input and then have an updated design on the spot.

18:27  
If it's a small thing like, Oh yeah, OK, we've got these tables wrong slightly, then sure, you can annotate it on the day.

18:35  
So data model is something, yes, you could be involved in requirements captured to some extent.

18:41  
Normally when you're a junior, somebody will be looking over your shoulder, right?

18:45  
And so you, you're less likely to make a mistake.

18:49  
When you are more senior, you'll be trusted with this.

18:54  
When you're less experienced, you will have normally somebody looking over your shoulder.

18:59  
But you do have to evaluate it, you know, sort of, I guess it's a reflective thing.

19:04  
You need to check yourself.

19:06  
Don't just go, oh, yeah, we got it all, now get on with it because the client might not be right.

19:11  
And you need to kind of ask, is this going to work?

19:14  
Will this hold all the data?

19:16  
So you remember in the discussion of the user interface design, we were saying once you produce the user interface design, it then becomes clear that some of the requirements haven't been documented.

19:26  
And in fact, the guest lecturer said exactly the same thing because this is what happens even if you chat to people, you don't have them all down.

19:35  
And then they say, oh, we need this feature.

19:37  
And when when you then put that into action, you say, oh, there's no requirement.

19:44  
So the same can happen with the data model that you put together what you think is the data model.

19:50  
And somebody says, oh, but you haven't considered these data and you're they're not in the requirements or in the user interface design.

19:57  
Oh, right.

19:58  
Yeah, we should definitely put that in.

20:00  
And then you may end up going back to the user stories even and adding those.

20:04  
So you have to be very careful that at any stage you can collect more by, you know, exposing more of the thought process.

20:12  
When we come on to this next section with the architecture design, it's normally done by a specialist.

20:19  
So somebody who's has more experience.

20:22  
There is actually a role called a software architect, or depending on what they're building on, you can have like a cloud architect who's specialised in designing the architecture of bits of software.

20:36  
And that person has not gone straight to that role.

20:39  
They've been a developer for a while, seen a few big projects and then they might do some additional training and then sit at that role.

20:46  
But if you screw up the architecture, which is our next section, bad things happen.

20:50  
So I remember 1 project which basically failed because the architecture was wrong and it didn't matter how much time you put in, it stayed failed because what was underneath was wrong.

21:04  
All right.

21:05  
Anybody else got any questions about data model before we go on?

21:15  
I don't know the term environment architect to be quite honest.

21:18  
A software architect or a is somebody who thinks about the big structures of the software environment Architect.

21:27  
I'm not sure what it means.

21:28  
I'll have to look it up.

21:31  
OK, so architecture, design patterns.

21:37  
So architecture is important because if we create the wrong thing, our data can't flow through it.

21:45  
Imagine if you decided you're going to have a server and you had brilliant code running in that server, but a server is only as good as that computer it's running on.

21:57  
So you can have a very, very, very powerful computer, but it still can be overwhelmed by the Internet.

22:04  
So, for example, you could have a computer with maybe A10 Gigabit input into it.

22:10  
You could have very fancy processes in it, could have, I don't know, a few terabytes of RAM, So that's memory, but you can still overwhelm it by sending out a lot of requests.

22:23  
So you need to think about the architecture.

22:26  
Is it the right architecture for the scale of the application?

22:30  
So you need to think, all right, today my application may be small, but in say a few months or years or whatever, what will it look like?

22:38  
Will it look big?

22:39  
And so the architecture choice at the beginning is important.

22:42  
Now, another point is it can be coupled to the software framework that you're using.

22:49  
What do I mean is that if you go for a certain software framework, they tend to enforce an architecture model.

22:58  
And we'll see some of these later on in the talk.

23:00  
So if you think about building a desktop application with Windows, they enforce a particular architecture.

23:07  
So by implementing in that framework, it actually is enforcing a architecture approach, which may or may not be a good one depending on what you want to do.

23:17  
So there's this sort of decision about architecture and technology and technology and architecture.

23:21  
It's somewhat self referencing.

23:24  
I'll come to some of the problems with the the potential forced architecture choices later on.

23:32  
All right, so architecture design patterns.

23:35  
Now these are established designs which you can combine one or more of into application.

23:43  
They're a bit like my analogy earlier about constructing a building with steel and then attaching stuff on the outside.

23:50  
They're things other people have done before and they work, and there could be documentation about how they work.

23:56  
So potentially they could be useful for your development team because everybody knows where they say bricks go on the answer of the building.

24:05  
In the same way, everybody knows where the functionality should go in the architecture because they've seen the pattern before.

24:12  
It also potentially leads to testing, which could be easier.

24:16  
So you've separated out pieces of functionality into different bits of code so people know how that is going to be used in terms of your development team.

24:27  
However, warnings, as I've already said, that architecture on its own can cause a project to fail, which is deeply embarrassing.

24:36  
If you spent a lot of money on it, it just doesn't work.

24:40  
So you can have problems, as I said, with a client server, it can be other things that you might think, oh, nobody ever does this, but yeah, they do.

24:49  
It's typically the combination of the client knows what they want and potentially the software builder believes maybe might work, but maybe doesn't have experience and agrees with the client.

25:06  
That's typically how it goes.

25:08  
And then you need somebody who's more senior to come in and say no, don't do that.

25:13  
It will cause you trouble.

25:15  
So if you get it wrong, it may be just slow.

25:18  
So it might just work or it might be overwhelmed easily, or you might not be able to expand it.

25:24  
Meaning you could have 100 users today and tomorrow you might have, I don't know, 10,000 or 100 million or whatever it is, right layered.

25:32  
This is the simplest architecture pattern and you can combine it with some of the others.

25:38  
The idea here is that we put our functionality into the layers.

25:44  
So we have a higher level layer which is closer to the user, and we have a lower level layer which is closer to the data storage or could be the database.

25:56  
Here it is in illustration form.

25:59  
You can have as many layers as you want.

26:01  
All right, so this one has four layers.

26:04  
At the back.

26:04  
Here we have a storage layer, and this appears to be some relational database layer judging by the illustration.

26:14  
And then above this we have a persistence layer.

26:16  
Now we'll come back to this with some code examples.

26:19  
Normally what you do when you're developing is you don't construct the database and the code separately.

26:26  
They might teach you this in the database module, but people don't do that.

26:31  
Normally what happens is you have what's called an Object relational mapper or object document mapper, and that generates the database from the code, or it can generate the code from the database and you tend to use this mapper in the persistency layer.

26:49  
So the persistency layer is the piece of code that is reading data from memory, writing to database, writing to a file or vice versa.

27:00  
So it's a layer which of which the job is simply to store and recover data.

27:06  
That's all it's job.

27:08  
So you don't put user interface functionality in here.

27:11  
You just put in here what is needed to store and recover the data.

27:16  
Now on top of that you might have a so-called business layer.

27:20  
Now this is often doing some real work, like it could be doing some calculations.

27:25  
Imagine you've asked your web service to calculate for you the thermal loss of a building.

27:31  
That would be in the business layer.

27:33  
It would be sat here.

27:34  
So the data have been read into memory by the persistency layer.

27:38  
The business layer does some calculations on it, and then it might hand them to the presentation layer.

27:44  
So the presentation layer is the layer that the user looks at, so the data expressed here.

27:51  
So the job of the presentation layer is to receive data and manipulate it so that the user knows what they are, and likewise to receive data from the user, validate them to check they're OK and then pass them on to the business layer.

28:09  
You can read about this architecture with the.

28:12  
I think it's a free book I attached to the myplace page.

28:16  
Here is the same setup but with closed layers.

28:20  
So you can have closed to open layers.

28:22  
The idea with closed to open is if it's closed the request from the top needs to \*\*\* through each of the layers.

28:31  
So here you can see that by the arrows here.

28:34  
So the user makes a request and that request is sent to the user interface and then to the business layer and then to the business layer and to the storage.

28:43  
So the user cannot directly access the persistency layer.

28:47  
In this case it's forbidden.

28:49  
That's what we mean by closed.

28:51  
You can also have a layer that's open.

28:54  
And so you can allow in some cases requests to bypass a layer.

29:00  
Now, this may well be perfectly reasonable.

29:03  
So here we've got a service.

29:05  
So this service layer is doing something else.

29:09  
Maybe it's looking up something about the client.

29:13  
It could be a data catalogue layer, a different web service.

29:18  
And so the request comes into the business layer and the business layer could go to the catalogue and get some more data and which would then go to the persistency layer and so forth.

29:28  
Or the business layer could just go and get the data it needs because it's just simple data straight to the persistency layer.

29:35  
So when you're thinking of a layered architecture, you need to ask yourself, do we want closed or open layers?

29:42  
Often you need closed layers and there isn't much reason to do what I'm showing you here.

29:47  
But sometimes there is and that's why I'm showing you.

29:50  
Is a concept.

29:53  
OK, client server.

29:57  
This is a very common one.

29:59  
What happens is that you decide you're going to put some of the functionality into the client.

30:04  
Now at the extreme level, the client could just be a connection to your server.

30:09  
So you log in and you send data could be from the command line to the server.

30:16  
The server does all the work and then returns an answer.

30:20  
So this would be like an old style telnet client.

30:23  
The client's doing almost no work.

30:25  
It is sending data to a server, which is then replying.

30:29  
Now you can choose how much do you want to do in the client and how much do you want to do in the server.

30:35  
As we've already said, the problem with servers is that they can be overwhelmed by lots of requests.

30:41  
So it's probably a good idea to think, can I put some of the functionality into the client?

30:48  
Because then if I have lots of clients working, that functionality will be running on lots of computers.

30:54  
So we've distributed the workload examples of this.

30:59  
We have e-mail clients.

31:02  
The e-mail server does do a lot of work, but with the modern e-mail client it is supporting the editing of the e-mail, the sending of the e-mail, maybe the checking of the grammar, or anything else.

31:15  
That's all in the client, and the server's job is basically just to send and receive the e-mail.

31:21  
A web browser is another example.

31:24  
A web server receives and sends data which could be HTML, so hypertext market language.

31:33  
It could be sending Jason which actually the client is rendering.

31:40  
So often modern applications run JavaScript or TypeScript in the web browser and that original code is download from the web server.

31:51  
And then what happens is the TypeScript of JavaScript is sending data to the server as Jason mostly these days, and then it is actually performing some of the processing locally.

32:04  
So when you see this, it is normally called a single page application because the data are downloaded into a web browser and it looks as though you're visiting different web pages, but you're not.

32:16  
What's happening is the web page URL is affecting the JavaScript TypeScript that's running locally and causing you to go to another page inside the single page web application.

32:30  
So with web applications, you can choose how much have you got in the server, how much have you got running in the browser.

32:38  
Now a server is a server because it listens and all servers, it doesn't matter which operating system, if it's Windows or Linux or Mac, doesn't matter.

32:50  
The server listens on a port.

32:52  
Now you have an Internet connection, which is often an IP connection, and that is associated with the computer.

33:02  
So my computer has one for example, because it's connected to the network.

33:06  
Now on my computer I might have a file server, and the file server is listening on a particular port number.

33:13  
So when an external connection comes to this IP address associated with my computer and asks that port number for a connection, it then creates a connection for that incoming client.

33:26  
So it keeps listening and it hands off the connection to the incoming client.

33:32  
Now you need to think about this carefully because each connection is a parallel process.

33:39  
So if you have lots of incoming connections, say for example a web server, what normally happens is the web server keeps listening on the same port and everybody comes in and asks for a web page.

33:51  
It gives them a separate connection.

33:53  
Now, if you keep giving people separate connections, you'll run out of memory and computer processing power.

33:59  
But memory's the more problematic issue because you can't have all these parallel threads running all at the same time.

34:07  
So what you'd normally do is you normally have a limit to the number of parallel threads that you can have processing at once.

34:14  
You have a thread pool, meaning you could have, I don't know, potentially a few 100 or a few thousand of them depending on what the server is, and then you recycle them.

34:23  
Now you recycle them normally by deciding the client's finished with this thread.

34:28  
So it could be a timeout.

34:30  
It could be OK, they're not connecting anymore, Fine, we'll drop it.

34:34  
It could be that you also have a block where you just don't hand out any more threads because your server's overloaded.

34:40  
I remember the one student one year had a very foolish application that kept opening and closing the connection to the database at such a rate the database couldn't recycle, the application couldn't recycle the connection threads and so nobody in the lab could do any work because they were just open and closing this connection to the database server.

35:03  
So you need to think about how do you recover from these scenarios and sometimes maybe you don't, maybe just fail nicely.

35:12  
Here's a web application in terms of a layered 1.

35:16  
So you have a web browser at the top here and inside the web browser could be running React.

35:22  
React is a JavaScript framework which you can build, use interfaces in.

35:28  
So this could receive a user request, send the request probably is some kind of web request, maybe with a bit of Jason to a web server, which here is implemented with Flask.

35:42  
Now we could implement the web server with other things.

35:45  
So we could use Node JS, wecoulduse.net, we could use Ruby on Rails, blah blah blah.

35:51  
Doesn't really matter.

35:52  
This is just one application that works and I can tell you about just now.

35:57  
Flask itself can read and write data from the database using an object relational mapper called Flask SQL Alchemy.

36:05  
So that has the job of saving and recovering data from a relational database sat at the back here.

36:12  
Now depending on the language we're using the framework, this will be a different piece of technology that we're using.

36:19  
Same function doing the same job but just different package, right?

36:27  
Another architect you can have is a so-called publish and subscribe.

36:32  
So this is where you are broadcasting out a message.

36:37  
Imagine you've got a computer sitting on the network and it says, oh, I've got a problem, and it doesn't know who to send that problem to.

36:45  
It just sends the problem across the local network.

36:49  
And then other computers on that network say, oh, the computer over there has a problem.

36:54  
That would be a broadcast message.

36:57  
Now you can have broadcast messages on the Internet or on a subnet, but you can also have them within an application, meaning pieces of your application are working together.

37:07  
One of them says, I want to send you some message data and the other one then can read.

37:14  
So it can be a listener.

37:15  
It can be all right.

37:16  
Yeah, I've got a little bit more data.

37:18  
So normally when you're talking about this, it's publish and subscribe.

37:22  
So somebody sending out the data and the other application is somehow associated with that message bus.

37:28  
So the data are coming through the bus and being collected by the other one.

37:35  
Now this, this particular terminology, message bus is an analogy to a computer or an electronic system.

37:45  
Now what you have with a computer, I'll go to my laptop just to illustrate the, the nomenclature is that we have a USB application port here, right?

37:56  
USB is a bus.

37:57  
We can send data over it.

37:59  
So it could be in this case audio data or something else.

38:04  
Now in an application, we have the same concept.

38:07  
We have a bus.

38:08  
Now it's not necessarily a physical bus, but it's a way of the data being streamed through the application, same as the electronics version.

38:19  
So we want in this bus some kind of memory so that data coming in.

38:25  
We keep the data for a little while and the subscriber can read them.

38:30  
This is really helpful when we've got distributed applications.

38:34  
So you've got different application parts and you don't want them to be tightly coupled to each other.

38:39  
So you don't want one sitting there waiting for the other one because that will slow the whole application down.

38:45  
So this is 1 bus that you can download and play with.

38:49  
It's called Rabbit MQ.

38:51  
It's probably reasonably well known.

38:54  
If you go on to the cloud, you'll find that Azure and AWS have their own buses that they've used or modified from somewhere else.

39:03  
There are several versions of this.

39:05  
Now in this case, what happens is somebody else has written the technology that manages the bus.

39:12  
You write the application that connects to the bus.

39:17  
So here we go.

39:18  
This is a bunch of different things you can do with Rabbit MQ.

39:22  
So you could have one application, it's sitting there sending out messages like somebody put the new electric shaver into the shopping basket, for example, that goes in the bus and then the consumer is consuming those messages and go, Oh yeah, that's client 84.

39:40  
We remember that for later.

39:42  
And so then when client 84 asks for something else, the Amazon or whatever says, would you like these electronic razor things?

39:50  
Because it's collected the messages from the the shopping basket application part.

39:57  
Now Rabbit MQ and the others will allow you to split out the messages.

40:01  
So for example, this is a publisher that's sending messages.

40:04  
The red boxes, by the way, are just memory buffers.

40:08  
So it's sending out the data, but the boss is accepting the messages and it's holding a certain number of them before it has to forget them and the consumer is reading them in order.

40:21  
You can have one to many with a split, or you can have these very complicated ones where you've got different pieces of data being streamed to different buffers.

40:30  
Anyway, the different bus technologies allow you to decide what you want to do.

40:35  
I'm only suggesting it as something to be aware of because potentially creating an application that is a publisher and a consumer like this is quite useful.

40:47  
Now, microservices includes the idea of a web service and often includes the idea of a publisher and consumer, which we've just been talking about in the previous slide.

41:00  
The idea here is if you have one web service and it is listening to incoming messages, it is just a very big BLOB of code.

41:10  
That's one problem.

41:11  
The other problem is if you want to accept more requests, it's hard to just run several copies of the same web server because the web server contains everything.

41:22  
Now imagine you had decided, ah, we've got a web server, but we're going to take part of it, which is the catalogue, the product catalogue.

41:30  
And that's going to be now a separate web server.

41:32  
And that one, if somebody wants to ask more and more questions of a catalogue, we're just going to run several more copies of the catalogue web server.

41:40  
And so that's essentially what the microservices model is.

41:44  
We want to think about the features within a web service and potentially we end up splitting what could be a large web service into several smaller ones because you can then run several instances of the smaller ones depending on what they're doing to allow more customers to use your web service.

42:05  
So the micro service application is potentially harder to build to start with.

42:10  
It's more complicated, but if you want to scale the application, meaning horizontally, several copies of it running at once on the cloud, it's very useful.

42:21  
Big web applications on the Internet servicing lots of customers typically are these days microservices.

42:29  
If you've got a smaller user base, it might not be worth doing.

42:32  
You can start off with a big web application and then you can factorise it into microservices.

42:39  
As you go, you can migrate.

42:42  
So this part of the architecture design is not too painful if you get it wrong all right.

42:49  
This is the so-called monolithic where people talk about it as monolithic because it's just one big web service.

42:56  
So what happens here is you have some clients and they send and receive messages to the server, which, yeah, sends and receives data normally to a database.

43:06  
So it's just one web server.

43:09  
Now when you go to the microservices model, what you normally have is some kind of application gateway.

43:16  
So what happens is the the clients send messages to a computer process and it looks like you're actually talking to one server, but you're not.

43:26  
And depending on what you want, that request is forwarded to different microservices.

43:32  
And normally these microservices have one database associated with them.

43:39  
Now, if you've got a microservice, you often have a bus in here.

43:45  
So I've put here the symbol for the Azure bus and Rabbitmq.

43:50  
So in this case, you've got two microservices running.

43:53  
They're both talking to their own database.

43:55  
They could talk to each other directly without a bus.

43:59  
Now what does this look like?

44:00  
This looks like a web service asking the other one for some data.

44:04  
Now it's going to sit there waiting until you have either the data or a timeout.

44:10  
Now this is exactly like you opening a web browser and going to a web page and waiting for it to load.

44:16  
And if it doesn't load, the web browser says server took too long to reply or something like that.

44:22  
So that's possible, but you might not want tight coupling between your web services.

44:27  
You might just want to put the message into the bus and the other one picks up the message from the bus.

44:35  
Here is a realistic scenario, one that you often see in textbooks, that is an online shop.

44:42  
And what happens here is that you have the application gateway.

44:48  
So somewhere over here is your web browser sends a message to the gateway.

44:53  
For example, can you please put my electronic shaver into the Amazon basket?

44:57  
Great.

44:58  
Now the basket itself stores what you've got in the basket in its own database.

45:05  
So there's a database here which is just used for that.

45:08  
So imagine that you shut down your computer, you went shopping, you came back, you fire up your computer again, you look in your basket and it's still there.

45:16  
It's still there because it's been stored probably in a relational database at the back here.

45:21  
You, as I've already said, more than likely want the product catalogue to be separate from the shopping cart because this is normally an application part where you're reading data.

45:31  
So you can have many product catalogues.

45:34  
So whenever anybody looks up data, they're going to go to a different instance because they are just providing the same data in parallel.

45:44  
And then you might want to recommend here.

45:47  
So the recommender's job is based on your previous ordering history.

45:51  
It suggests new things.

45:53  
Now, there's no reason for the recommender to be tightly coupled to the user's request.

45:59  
So if a user says put the electronic shaver in my basket, the user doesn't need to wait there for the recommender.

46:06  
So the recommender can be quite happy to find the data in the bus.

46:10  
Oh look, the user's bought the electronic shaver.

46:13  
Great.

46:14  
I'll store that in my recommendations data and when a search is next done, we'll be able to provide some recommendations to the user.

46:24  
All right, the benefits of microservices, each microservice is independent.

46:29  
Now you can implement them in different programming languages.

46:33  
So you could implement one in Ruby on Rails, one in Node JS, another one in pythonanotheronein.net, so on and so forth.

46:40  
You normally don't want to do that too much just because it's going to make your life harder.

46:45  
But there could be one or two of them that you want to do a different language and that's fine because you talk to them using a standard interface.

46:53  
So it's a web service interface or it's a bus interface.

46:56  
So it doesn't actually matter what the programming language is inside.

47:00  
You can update one of the microservices and not update the rest.

47:03  
So if you think about the monolithic one, when you roll in an update, you're updating the web server, whereas if it's a microservice, you're just updating one part.

47:12  
It's easier to scale because you can run several instances of a microservice much more easily than somehow duplicating a monolithic one.

47:22  
Yeah, it's more flexible.

47:23  
So you can even go as far as having a completely different operating system on your microservice.

47:28  
Like you could have one running Windows and the rest running Linux if you wanted to.

47:34  
The issues you need to have more experience in terms of the architecture design.

47:41  
How do you split into the microservices?

47:44  
It's something of a skill to know how to do this.

47:48  
Testing is a bit more of a pain because you've got to test all these microservices together and check they all work together.

47:55  
So yeah, testing is more complicated.

47:57  
Once you've set this up, it becomes easier.

47:59  
But to start, setting it up is hard work.

48:02  
Deploying it is a bit more complicated.

48:05  
The connections between the microservices have to be secured often, and keeping the security between all the microservice elements adds another layer of complexity.

48:16  
So you need to decide, do I need to do this or not?

48:20  
There is a gain, but there's also a penalty.

48:23  
You have latency.

48:24  
Now latency is simple.

48:27  
I ask a question and I have to wait a little while before I get an answer.

48:31  
Now, if you have a lot of web services, web services don't respond super fast.

48:37  
It might take a fraction of a second to respond.

48:39  
Whereas if it is a process call inside a web service, that might occur in a few milliseconds.

48:46  
So yes, you have latency, but generally speaking, if you have a large number of requests and you spread it across a large number of microservices, latency actually doesn't matter.

48:58  
But you need to think about where this latency occurs.

49:01  
And that goes back to thinking about how do you decouple the pieces into microservices, because if you split them in the wrong place, you have a lot of requests between them and you'll have a lot of latency problems.

49:14  
All right, so that's microservices.

49:16  
Does anybody have any comments or questions about them before I go into something else?

49:25  
No.

49:25  
OK, so microservices are a bit more complicated and maybe your brain's already going, can't cope.

49:33  
Functions of service is a simpler thing which is rather like a microservice.

49:38  
The idea here is we just have one function and it's operating as if it's a web server, and it normally is a web server and we can send it one type of request.

49:49  
Now this is quite pleasant to use because on a cloud you can run this function as many times as you like.

49:58  
So imagine I have a piece of code that receives some data from a weather station and cheques the data.

50:04  
Now the cloud will allow me to run this function and it will automatically create a multiple number of these functions to service my requests.

50:13  
So if I put together the functions service deployed to the cloud, I can then have a well, literally thousands of weather stations all sending requests, It appears to the same function as a service, but it's not actually, it's a copy.

50:28  
So it's, it's a bit like the microservice, but we've made it as simple in terms of its functionality and we can therefore clone them.

50:38  
And normally the cloud is doing this for us.

50:42  
They often receive HTTP messages which are web requests, but not only they can be activated on data storage changes, like maybe the database is updated and then the functions of the service wakes up and does something.

50:55  
They can return web requests themselves, so some data back to your web client, or they can save some data to storage.

51:05  
Now the benefits of function of the service is that they are smaller than a micro service.

51:11  
Normally the cloud that supports them does all the cloning for you.

51:17  
So you just need to produce a tiny bit of code, put it in a box, deploy it to the cloud.

51:23  
They're very lightweight and often the cloud people charge you once per request of the function, and the charge per function call is normally tiny.

51:33  
However, if you call it a lot, yeah, does not 'cause you a real cost mast and slave now.

51:44  
It's the old way of talking about it.

51:46  
I don't know, they might have come up with a political orderly correct version of this.

51:49  
It's nothing to do with slavery.

51:51  
It's just the way the IT people used to call this.

51:55  
Now the idea with the master and the slave is that you have a web server or a server or some sort and one of them is in charge.

52:04  
Imagine you've got a relational database.

52:07  
One relational database can accept write requests, the other one accepts copies from the first one.

52:15  
Both of them accept requests for data.

52:18  
Now you have to do that when you duplicate relation databases, because there needs to be one point where the data are written.

52:25  
So you have one master and potentially many slaves.

52:28  
In other services you can do the same.

52:32  
For example, if you ask who is this on the computer network, that request may be sent to a master process, and if the master process knows the answer, yeah, it'll give you the response.

52:44  
If it doesn't have the ability to answer, that process request might go to a slave.

52:50  
So sometimes your architecture is clever, meaning I turn off the master server and then the slave becomes the master and then everything goes on as usual.

53:03  
So when I was working in a different department, I was managing server and master and slave configurations, and we had these running and it was so good you could turn one off.

53:15  
Nobody noticed the other one became in charge.

53:18  
You turn the oil back on, nobody noticed.

53:21  
It would just swap the load between the two.

53:26  
Right?

53:27  
peer-to-peer is another one.

53:28  
This was really popular in the bad old days where people were pirating software and things and sharing them around.

53:36  
There are good reasons for having peer-to-peer.

53:39  
You remember the client server architecture.

53:42  
The problem there is no matter what you do, eventually that server is overloaded.

53:47  
So with peer-to-peer, there is no server.

53:50  
You essentially have a series of things on a network which have a shared responsibility.

53:57  
Now the problem with this, well, yes, it's great in terms of resilience.

54:01  
If you knock out and node, it doesn't matter.

54:04  
But it's much more complicated to configure who's in charge.

54:08  
How do we trust people on a network that is peer-to-peer?

54:13  
So you it's necessary that you implement logging and other kind of security mechanisms which are rather more complicated.

54:20  
So if you want real resilience, peer-to-peer might be for you.

54:24  
But on the downside, the security aspects are going to be a lot harder.

54:30  
Now peer-to-peer applications, where do you see this in use?

54:33  
Internet of Things devices.

54:35  
So for example, if you deploy a device that is measuring how a how the supply is working, there might be many of those on a small scale network, which is a mesh network.

54:47  
There is no hierarchy there.

54:49  
They're sending data around them.

54:52  
For example, electronic light bulb that's Internet of Things is connected potentially to a network.

54:59  
Now you can have spoof and replay attacks.

55:02  
I don't know if Rose has told you about these, but anyway, what happens is you pretend to be a valid device on a network.

55:10  
So, for example, you can pretend to be a light bulb that and you say, oh, honest, I'm a light bulb.

55:17  
You connect to the network and then you're in the mesh network and you do bad things.

55:23  
Yes.

55:24  
Where do you see this used messaging and communication, file sharing.

55:29  
Yeah.

55:29  
And as I've already said, that was associated with software piracy because because it's not centralised, nobody can come in and shut it down, right?

55:38  
So that's why pirates like it.

55:42  
All right, So stepping away from the more generic models, and now we're going towards specific models.

55:48  
So this one is often used for web services.

55:51  
It's probably less used now than it was.

55:54  
It's called model view controller.

55:58  
So what we have here is our view is normally the data that are being shown to the user.

56:05  
It could be shown directly to the user or it could be shown and then rendered by the web browser.

56:12  
The controller receives the web request, so any request for a web page or something goes to the controller and then the controller manipulates the data model.

56:22  
So it takes some data, does something with it and sends it back to the view.

56:27  
So this is model view controller.

56:28  
Now notice you don't have a persistency layer in this diagram, because when you have model view controller in a real web application, you actually have more than one architecture pattern.

56:40  
You have this model view controller and then you normally have a layered application for the database part.

56:47  
So it's a combination of the two architecture patterns.

56:51  
This originally came from small talk, which you probably don't know about, maybe do.

56:55  
It's an old programming language which people don't use that much anymore, but has been used a lot in Microsoft's web application framework.

57:05  
It's less popular now because although it allows you to separate things out potentially, it slows you down in the way that data are passed around the layers.

57:17  
So what happens with model view controller?

57:20  
The user views the data through the view either directly or it's rendered in the browser and then viewed.

57:27  
Data requests go to the controller.

57:29  
You can have more than one controller, so you can have a controller associated with different parts of your web application.

57:37  
And then the controller has all the code in.

57:39  
So the functional code.

57:41  
You can put some code in the model layer, which could be just to calculate the transient things.

57:46  
And you can put code, a little bit of code in the view layer, but that's more about checking the inputs or how it's displayed.

57:57  
MVC applications, yeah, normally web applications, not always.

58:01  
And you see this often with Microsoft, as already said, so.net or C#, that's Microsoft's web application framework.

58:12  
You when you're coming to test this, you can just test the controller.

58:16  
That's the nice thing.

58:17  
Test the controller separately, worry about the views.

58:22  
It tends to be slower than not using MVC because of the data being passed between the layers.

58:29  
And as I've already said, you have a database which is normally a separate architecture pattern.

58:35  
If you're using Microsoft, you can use Razor on the server side.

58:41  
So Razor is a way of implementing the view and then it turns that into HTML, which is then shown in the client side.

58:50  
Or you can use what we were talking about before, which is a single page application and you send Jason back to the single page web application.

58:59  
So this view could be Razor based, it could be Jason based.

59:03  
It depends on the framework that's producing the model view controller implementation in mobile applications.

59:09  
You see this with Flutter.

59:11  
So Flutter will allow you to implement model view controller.

59:15  
Flutter is a framework for implementing mobile applications that are portable platform independent.

59:23  
So you can run them on Mac or on Android.

59:28  
On iOS, that's the platform development for Mac mobile phones will allow you to implement MVC.

59:38  
Now model view presenter, I'm mentioning because it's still used and it's a stepping stone between model view controller and another architecture pattern, which I'm going to talk about in a minute.

59:47  
It's a generalisation of model view controller.

59:52  
Now what happens here is the view is passive, meaning it doesn't do anything.

59:56  
It's just sort of stick it on the screen.

59:58  
There's no active code here.

1:00:01  
In this model, the controller is actually called the presenter, although the idea is similar.

1:00:06  
It is replying, doing things, and then the model itself is the data that are held in memory.

1:00:14  
Again, there's no database here because you would need another architecture for that, a layered one.

1:00:20  
This originally came or originally mentioned from the publication at the bottom here.

1:00:25  
This is 1996.

1:00:29  
Now the further specialisation of the Model View presenter is the so-called Model View View Model.

1:00:36  
Now you think who comes up with these things right Microsoft.

1:00:41  
To blame here I would argue is their acronym.

1:00:45  
So this was developed by Microsoft for desktop applications.

1:00:49  
Now what people do with user interface applications is they come up with an easy way of describing how the components are on the screen.

1:00:58  
So like where is my window?

1:01:01  
I don't really want to have to code that up.

1:01:02  
I want to drag them in and then maybe these are described using XML or something else.

1:01:08  
So often what happens is the the the view is actually not normal code and then the the part of the application that does something needs to be somewhere else.

1:01:20  
So this is actually similarish idea to the model view controller or the model view presenter idea that we're fragmenting it so that we can have pieces that we can easily implement using drag and drop.

1:01:34  
Anyway, it's used by Microsoft for desktop applications.

1:01:38  
It's also potentially used by Android applications as well.

1:01:43  
What happens here is the user again interacts with the view, so the view is what they see and they can be user interface elements that you drag and drop in.

1:01:55  
And then these view elements are connected to the view model and the view model is banned to the view through functions.

1:02:04  
Now, sometimes you hear things where programmers talk about callbacks or bindings.

1:02:10  
What this implies is if you do something like press this button on your keyboard, the computer is listening to the keyboard and receives that request and then does something.

1:02:21  
Now you can do the same in a user interface.

1:02:25  
You can have a binding between an application feature and a function like, oh, somebody's pressed the save button, now all the data are going to go into the function that's associated with the save button.

1:02:35  
So you're separating out the view from the view model.

1:02:40  
The active code is either in the view model or in the model.

1:02:44  
So you normally only have things which create transient data in the model.

1:02:48  
Mostly the functions that have been called are in the view model.

1:02:52  
Don't ask me why they call it view model, I can't remember.

1:02:55  
Doesn't seem too logical to me.

1:02:56  
But that's just what they call it.

1:02:58  
So here it is.

1:03:00  
We've got the view, and we've got these bindings where the view is somehow connected with functionality that's in the view model, so that when you do something in the view, it automatically calls the right function in the view model.

1:03:14  
And then the view model updates the model data, Some notifications come back back to the user, so on and so forth.

1:03:20  
You can read about it from the web page links to the bottom.

1:03:24  
So Model View Viewmodel is used by the Windows Presentation Foundation or WPF which is their desktop application framework.

1:03:36  
It's uses XAML markup underneath XML style markup for describing the user interface elements.

1:03:46  
And then the actual functions are implemented using C#, which is a bit like Java right now.

1:03:53  
As I've said all all the time, well several times in this lecture, you can combine many different pieces together and that might mean you've got an MVC and you've got a layered pattern, or you could have a master of slave which also has inside a layered pattern and you can have a message bus and that's fine.

1:04:17  
You need to distinct to yourself which of these models could be useful and then come up with a design.

1:04:23  
As as we've already said, it's normally down to a more experienced person to do this, but in the lab we're going to have fun and try and do it ourselves.

1:04:31  
OK, so conclusion of the lecture, we need to capture that data model early on because it affects the complexity of the whole application.

1:04:42  
We need appropriate architecture models.

1:04:47  
The architecture model idea is great because people start to get to know where things go.

1:04:51  
Oh yeah, got a new user interface feature that must go in the view.

1:04:55  
It's not a massive BLOB of code.

1:04:57  
You know, the plum pudding where everything's all over the place.

1:05:01  
People start to put things in the right place because it's part of the architecture pattern.

1:05:06  
It can be easier to test and expand depending on the architecture choice, of course, what you're doing.

1:05:12  
And as I already said quite a number of times, bad architecture choices can be very costly and cause your project to fail, which is bad and embarrassing if you do it.

1:05:23  
So don't do it.

1:05:24  
Anyway, that's the end of the lecture.

1:05:26  
Does anybody have any questions or comments, Thoughts you can ask me about real problems if you so wish, or anything else you'd like.

1:05:38  
I'm throwing a lot of things at you just to sort of give you lots of ideas.

1:05:43  
And it's probably a good idea to go away and think about them and come back in the lab and have a go at them and then go away and think about them again.

1:05:51  
The purpose of showing you all these things is that when you then come across a framework and somebody says, oh, we're using model, view, view, model.

1:06:00  
And you go, Oh yeah, I remember something about that.

1:06:03  
It's not strange or unusual.

1:06:06  
You've got a kind of starting point.

1:06:08  
Yeah.

1:06:09  
And I've given you some examples of when you'd use these models and how you might combine them together.

1:06:16  
All right.

1:06:17  
Anybody questions, comments, thoughts.

1:06:22  
Yes.

1:06:25  
Yes.

1:06:31  
With the layered architecture, you can do whatever you like.

1:06:35  
Yeah, you can have several layers open if you so wanted.

1:06:39  
So for the closed one here, this one, there could be some request coming from the user which would probably have to go through the presentation layer because the user hasn't maybe got the experience to go to the back end database.

1:06:57  
You could for an administrator have some kind of API where the administrator goes directly through some web interface and talks to the database.

1:07:06  
That would be like an administrative process that's allowing you to get to the database.

1:07:12  
You decide what you're going to allow.

1:07:14  
Basically.

1:07:16  
You probably want to keep things closed for most people, but there could be some reasons why you just want to give them the raw data or some kind of like high level role that needs to get to the back of the application.

1:07:30  
Yeah, but within these layers you often have cheques to do with the data that.

1:07:36  
So for example, the data that come in, you check them before you hand them on to the next layer.

1:07:41  
So if you miss out a layer, you may be missing out some of the validation.

1:07:45  
Maybe you don't need it depending on what you're doing.

1:07:47  
But yeah, so in short, you can open them all if you want, but you want to think about valid processes of your application and what you want to allow or not.

1:07:57  
Yeah.

1:07:59  
Anybody else?

1:08:03  
That was a good question.

1:08:05  
There must be some burning questions.

1:08:10  
We have two hours because you can discuss, you see, but you have no questions.

1:08:14  
We can leave it there.

1:08:17  
All right, We can leave it there.

1:08:19  
Please come and ask in the lab or later or whatever, if any of these things are confusing.

1:08:24  
Have a read of the some of the references.

1:08:27  
They're all ideas for a reason, and if you know a little bit about them, you kind of see where they might be useful or not.

1:08:34  
OK, well, thanks very much.

1:08:36  
I'll hang around if anybody's shy.

1:08:37  
Otherwise, I'll see you hopefully in the lab.