

Session 7

Component and deployment diagrams

Component and deployment diagrams describe the physical side of the design: which pieces of software exist, what interfaces they expose, and which machines they run on.

Objectives

✕ Do not copy. Read for understanding and the viva

- Complete questions 16 to 18 of the manual: component and deployment diagrams
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

✕ Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q16	Draw Component Diagram for Online Examination System	Complete
Q17	Draw Component Diagram for Order Processing Application	Complete
Q18	Diagram Deployment Diagram for Online Student Admission System	Complete

Preparation

✕ Do not copy. Read for understanding and the viva

- The manual asks for a problem description of 300 to 500 words and a list of assumptions before every diagram. Write both first; they fix the scope the evaluator marks you against.
- Group classes into components by cohesion (UI, business logic, persistence) and show provided and required interfaces with the lollipop and socket notation.

- For deployment, name the nodes (client browser, web server, database server) and the artefacts placed on each, with the communication paths between them.

Question 16

Problem Statement

■ Write in lab record

Draw Component Diagram for Online Examination System.

Solution

■ Write in lab record

Assumptions

The Online Examination System conducts university examinations over the web. Students log in, see the examinations scheduled for them, attempt a paper against a timer, and later view their results. Faculty maintain a question bank and assemble papers from it. Evaluators mark descriptive answers; objective answers are marked automatically. Results are published and the student is notified by email and SMS. The system runs as a web application with a browser front end for each type of user, a set of server-side services, and a relational database.

A component diagram shows how the software is divided into replaceable parts and how those parts depend on each other through interfaces. It does not show classes or objects; it shows the units that would be built, tested and deployed separately. The classes from the Session 1 class diagram are grouped into components by cohesion. Everything about a running attempt (timer, saving answers, locking) goes into the Exam Engine. Everything about authoring and storing questions goes into the Question Bank. Marking, both automatic and manual, goes into the Evaluation Service, and the totalling, pass or fail decision and publication go into the Result Service. Login and role checks are one Authentication Service used by every front end. Sending email and SMS is a Notification Service that the Result Service and the Exam Engine both use. The two front ends, Student Portal and Faculty Console, contain only pages and controllers. The Exam Database component is the persistence layer and offers one data access interface to the services.

Each dependency between components is drawn as a named interface. The component that implements the interface shows it as a lollipop (provided interface); the component that calls it shows a socket (required interface). Naming the interfaces is what makes the diagram useful: the Student Portal does not depend on the Exam Engine as a whole, only on IExamDelivery, so the engine can be rewritten as long as IExamDelivery is kept. Three packages, Presentation, Application and Data, group the components into layers and make the dependency direction visible: front ends depend on services, services depend on the database, and nothing depends upward.

- The system is a single web application; front ends are browser pages served by the same deployment, not native apps.
- Authentication is internal (enrolment number and password), not delegated to an external identity provider.
- The Question Bank is a component of this system, not a shared university service.
- One relational database holds users, questions, attempts and results; it is drawn as one component with one interface.
- Email and SMS are sent through the Notification Service, which wraps external gateways that are outside the diagram.
- Proctoring, payment of examination fees and re-check workflows are out of scope.

Steps

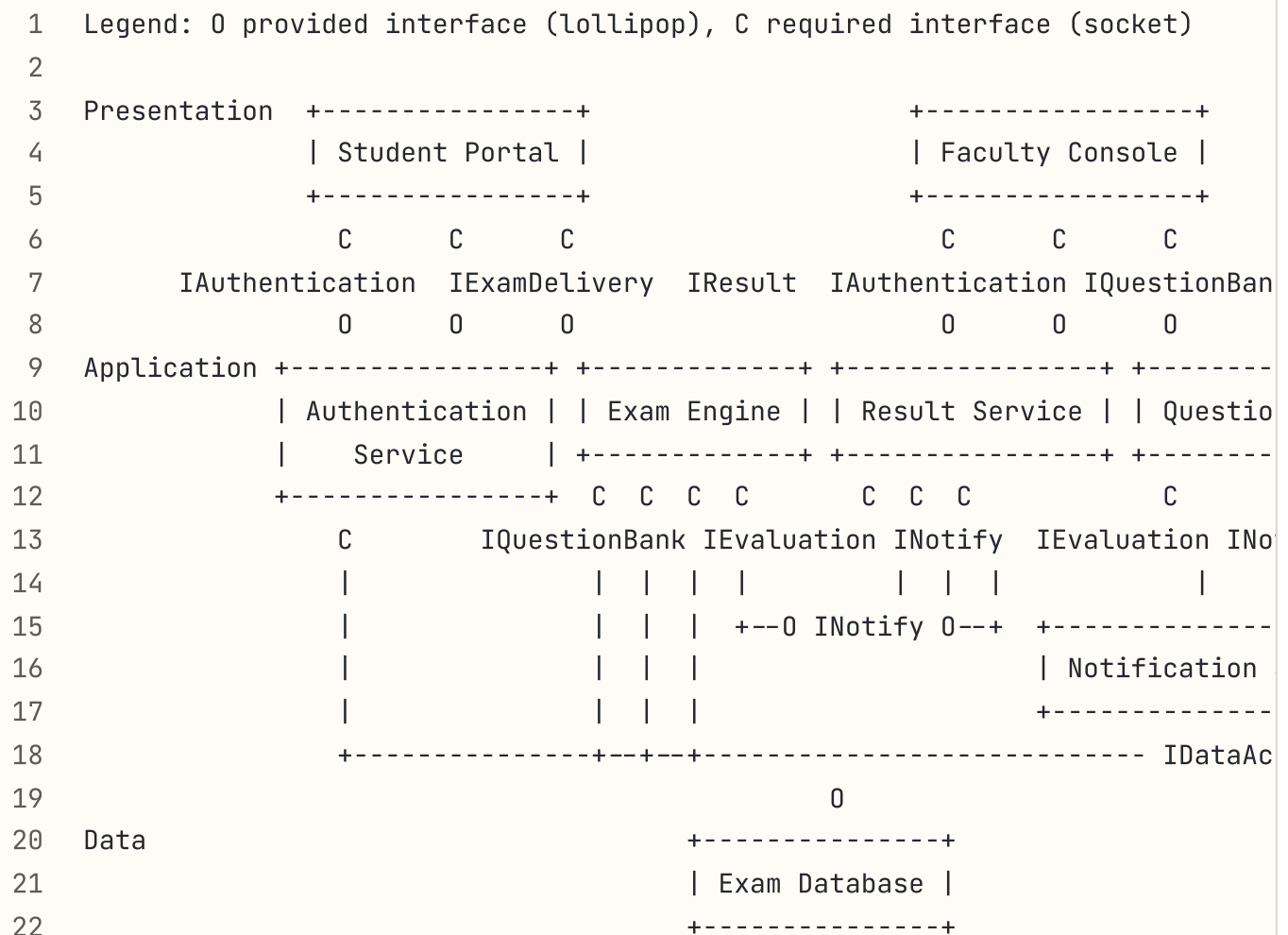
1. Choose Model, Add Diagram, Component Diagram and name it "Online Examination Components".
2. In the Toolbox choose Package and draw three packages named Presentation, Application and Data.
3. Choose Component and place the nine components from the table inside their packages.
4. Choose Interface and place one circle for each interface in the table next to the component that provides it.
5. Choose Interface Realization and drag from the providing component to the interface (StarUML draws the lollipop). Choose Dependency and drag from each requiring component to the interface (StarUML draws the dashed arrow; set the notation to socket in the Format menu if you want the half-circle).
6. Export the diagram as PNG.

Diagram

Component	Package	Provides	Requires
Student Portal	Presentation	none	IAuthentication, IExamDelivery, IResult
Faculty Console	Presentation	none	IAuthentication, IQuestionBank, IEvaluation
Authentication Service	Application	IAuthentication	IDataAccess
Exam Engine	Application	IExamDelivery	IQuestionBank, IEvaluation, IDataAccess, INotify
Question Bank	Application	IQuestionBank	IDataAccess
Evaluation Service	Application	IEvaluation	IDataAccess
Result Service	Application	IResult	IEvaluation, INotify, IDataAccess
Notification Service	Application	INotify	none
Exam Database	Data	IDataAccess	none

- Lab record: every tab is one file of the answer. Write all of them.

Sketch



Explanation

The diagram is read along the interfaces. A solid line from a component to a circle means the component implements that interface; a dashed arrow into the circle means the component calls it. So the Student Portal calls IExamDelivery, the Exam Engine implements it, and the portal never sees the engine's internals. The layering rule is visible: every dashed arrow points from Presentation into Application or from Application into Data, never the other way, which is what makes the front ends replaceable without touching the services. Two components require IEvaluation for different reasons: the Exam Engine triggers automatic marking when an attempt is locked, and the Result Service reads marks to compute totals. That is normal and is why the interface, not the component, is the unit of dependency. The Notification Service has no required interfaces because it only wraps external gateways; the Exam Database has none because it is the bottom of the stack.

Question 17

Problem Statement

■ Write in lab record

Draw Component Diagram for Order Processing Application.

Solution

■ Write in lab record

Assumptions

The Order Processing Application takes orders from customers of an online store and sees them through to delivery. A customer browses the catalogue, adds items to a cart, and places an order. The application checks that the items are in stock, reserves them, takes payment through an external payment gateway, creates a shipment with a courier, and keeps the customer informed by email at each step. Store staff use an admin console to manage products and stock levels and to look at orders. The application runs as a web application with a browser storefront, server-side services and a relational database.

The component diagram groups the classes of the application into deployable, replaceable parts and shows the interfaces between them. The Storefront and Admin Console are the two front ends and contain only pages and controllers. The Order Service is the heart of the application: it owns the order life cycle and coordinates everything else, so it requires the most interfaces. The Catalog Service owns products and prices; the Inventory Service owns stock levels and reservations; the Customer Service owns accounts and addresses; the Payment Adapter and the Shipping Adapter wrap the external payment gateway and courier API behind interfaces that the Order Service can use without knowing which provider is behind them; the Notification Service sends emails. The Order Database is the persistence layer with one data access interface.

The important design decision that the diagram records is the adapter pattern for external systems. The Order Service requires `IPayment` and `IShipping`; it does not require the payment gateway or the courier. The adapters provide those interfaces and are the only components that know the provider's protocol. Changing the payment provider means replacing one component. The external systems themselves are drawn outside the packages as components with their own provided interfaces, so that the diagram shows where the application's boundary is and which interfaces cross it.

- One store, one currency, one warehouse; multi-warehouse allocation is out of scope.
- Payment is taken through one external gateway over HTTPS; card details never enter the application, the gateway hosts the payment page.
- Shipping is arranged through one courier API; the application stores the tracking number it returns.
- Stock is reserved when the order is placed and released if payment fails; that logic is inside the Inventory Service.
- The Customer Service handles login, registration and addresses; there is no separate authentication component.
- Returns and refunds are out of scope.

Steps

1. Choose Model, Add Diagram, Component Diagram and name it "Order Processing Components".
2. Draw packages Presentation, Application, Data and External.
3. Place the components from the table in their packages and add one Interface circle per row of the provides column.
4. Draw Interface Realization from each provider to its interface and Dependency from each requirer to the interfaces it needs.
5. Export the diagram as PNG.

Diagram

Component	Package	Provides	Requires
Storefront	Presentation	none	ICustomer, ICatalog, IOrder
Admin Console	Presentation	none	ICustomer, ICatalog, IInventory, IOrder
Customer Service	Application	ICustomer	IDataAccess
Catalog Service	Application	ICatalog	IDataAccess
Inventory Service	Application	IInventory	IDataAccess
Order Service	Application	IOrder	ICatalog, IInventory, IPayment, IShipping, INotify, IDataAccess
Payment Adapter	Application	IPayment	IGatewayAPI
Shipping Adapter	Application	IShipping	ICourierAPI
Notification Service	Application	INotify	none
Order Database	Data	IDataAccess	none
Payment Gateway	External	IGatewayAPI	none
Courier System	External	ICourierAPI	none

■ Lab record: every tab is one file of the answer. Write all of them.

Sketch

```

1  Legend: 0 provided (lollipop), C required (socket), dashed = uses
2
3  Presentation      +-----+                +-----+
4                    | Storefront |            | Admin Console |
5                    +-----+                +-----+
6                    C   C   C                C   C   C   C
7                  ICustomer ICatalog IOrder    ICustomer ICatalog IInventory IOrder
8                    0     0     0                0     0     0     0
9  Application
10  +-----+ +-----+ +-----+ +-----+
11  | Customer Service | | Catalog Service | | Inventory Service | | Order Ser
12  +-----+ +-----+ +-----+ +-----+
13          C                C                C                C C C C
14          |                |                |          ICatalog IInventor
15          |                |                |                | |
16          |                |                |          IPayment IShippi
17          |                |                |                0   0   0
18          |                |                | +-----+ +
19          |                |                | | Payment Adapter | |
20          |                |                | +-----+ +
21          |                |                |          C
22          +-----+ +-----+ +-----+ +-----+ IDataAccess
23                                0
24  Data                        +-----+      Exter
25                                | Order Database |
26                                +-----+

```

Explanation

The Order Service is the coordinator, so most dashed arrows leave it: it prices the order through ICatalog, reserves stock through IInventory, charges through IPayment, books delivery through IShipping and informs the customer through INotify. None of those are the external systems directly. The Payment Adapter provides IPayment to the application and requires IGatewayAPI from the outside world, which is the adapter pattern in component form: the interface the application wants on one side, the interface the provider offers on the other. The External package draws the boundary of what the team builds. Components inside it have no required interfaces because the application does not know or care what they depend on. As in Question 16 the dependency arrows point only downward through the packages, and the database is the only component that every service requires.

Question 18

Problem Statement

■ Write in lab record

Diagram Deployment Diagram for Online Student Admission System.

Solution

■ Write in lab record

Assumptions

The Online Student Admission System lets applicants apply to a university's programmes over the web. An applicant creates an account, fills in the application form, uploads scanned certificates and a photograph, pays the application fee through a payment gateway, and tracks the status of the application. Admission officers log in from the university office to verify documents, shortlist applicants and record decisions. The system emails and texts applicants at each step. The application is a Java web application deployed on the university's servers.

A deployment diagram shows the hardware and execution environments (nodes) on which the software runs, the artefacts (the built files) placed on each node, and the communication paths between nodes with their protocols. The purpose is to answer the operations questions the other diagrams do not: how many machines, what runs where, and how they talk to each other. The layout here is the standard three tier web deployment. Client devices run a browser and hold no artefacts of the system. A web server terminates HTTPS, serves static files and forwards application requests to the application server, so that only one machine faces the internet. The application server runs Apache Tomcat with the admission web archive. The database server runs MySQL with the admission schema. Uploaded documents are large and are read rarely, so they live on a separate file store rather than in the database. The payment gateway and the email and SMS gateway are external services drawn as nodes outside the university's network. Each communication path is labelled with its protocol and port so that the network team can configure firewalls from the diagram.

The artefacts follow the component diagram of the same system: the web archive contains the presentation and application components, the schema is the data component, and the static bundle is the part of the presentation layer that needs no server processing. Devices are

stereotyped device and the software containers that run on them are stereotyped execution environment, which is the UML 2 way of separating the hardware from the runtime.

- The university hosts the system on its own servers; no cloud auto-scaling, one instance of each server.
- Clients use any modern browser over the public internet (applicants) or the campus network (officers); nothing is installed on client machines.
- The web server is Nginx and acts as reverse proxy and static file server; TLS is terminated there.
- The application server is Apache Tomcat 10 running one web archive, admission.war, built from the Spring Boot project of Section 2.
- The database is MySQL 8, reachable only from the application server on port 3306.
- Uploaded certificates are stored on a file server mounted over NFS; the database holds only their paths.
- Payment and messaging are external services reached over HTTPS; their internal structure is not modelled.

Steps

1. Choose Model, Add Diagram, Deployment Diagram and name it "Admission Deployment".
2. Choose Node and place the nodes in the table. Select each one and set Stereotype in the Editor to device or execution environment. Drag an execution environment node inside its device node so that it nests.
3. Choose Artifact and place each artefact inside its execution environment node; type the file name as the artefact name.
4. Choose Communication Path and connect the nodes as in the table. Select each path and type the protocol and port as its name.
5. Export the diagram as PNG.

Diagram

Node	Stereotype	Contains	Artefacts
Applicant PC or Phone	device	Web Browser (execution environment)	none
Admission Officer PC	device	Web Browser (execution environment)	none
Web Server	device	Nginx 1.24 (execution environment)	static bundle: html, css, js
Application Server	device	Apache Tomcat 10 (execution environment)	admission.war, application.properties
Database Server	device	MySQL 8 (execution environment)	admission_db schema
File Server	device	NFS export	uploaded documents
Payment Gateway	external node		
Mail and SMS Gateway	external node		

From	To	Protocol
Applicant browser	Web Server	HTTPS 443
Officer browser	Web Server	HTTPS 443 (campus network)
Web Server	Application Server	HTTP 8080, reverse proxy
Application Server	Database Server	JDBC over TCP 3306
Application Server	File Server	NFS
Application Server	Payment Gateway	HTTPS REST
Application Server	Mail and SMS Gateway	SMTP 587 and HTTPS

■ Lab record: every tab is one file of the answer. Write all of them.

Sketch

```

1  <<device>> Applicant PC or Phone          <<device>> Admission Officer PC
2  [ <<exec env>> Web Browser ]                [ <<exec env>> Web Browser ]
3      | HTTPS 443                                | HTTPS 443 (campus)
4      +-----+-----+-----+-----+
5                      |
6      +-----+-----+-----+-----+
7      | <<device>> Web Server                    |
8      | <<exec env>> Nginx 1.24                  |
9      | artifact: static bundle                |
10     +-----+-----+-----+-----+
11                      | HTTP 8080 (reverse proxy)
12     +-----+-----+-----+-----+
13     | <<device>> Application Server            |
14     | <<exec env>> Apache Tomcat 10            |
15     | artifact: admission.war                |
16     | artifact: application.properties       |
17     +-----+-----+-----+-----+
18     JDBC TCP 3306 | | NFS | HTTPS | SMTP 587 / HTTPS
19                  | |    | REST  |
20 +-----+-----+-----+-----+
21 | <<device>> | | <<device>> | | Payment | | Mail and SMS Gateway |
22 | Database Server | | File Server | | Gateway | | (external) |
23 | <<exec env>> | | artifact: | | (extern) | +-----+
24 | MySQL 8 | | uploaded | +-----+
25 | artifact: | | documents |
26 | admission_db | +-----+
27 +-----+

```

Explanation

Read the diagram from the clients downward. Nothing of the system is installed on the client devices; the browser is an execution environment that receives the static bundle over HTTPS at run time, which is why the client nodes hold no artefacts. The web server is the only node exposed to the internet: it terminates TLS on port 443 and forwards to Tomcat on port 8080 over the internal network, so the application server, the database and the file server can sit behind a firewall that admits only the paths drawn. Each communication path is labelled with the protocol because the diagram is the specification for those firewall rules. The application server holds two artefacts: the web archive, which is the compiled form of every component in the component

diagram, and the properties file, which is the deployment-specific configuration (database URL, gateway credentials) kept outside the archive so the same archive can be deployed to a test server. The external gateways are drawn as clouds with no internals because the university neither owns nor deploys them; only the protocol used to reach them matters here.

Viva Questions

✕ Do not copy. Read for understanding and the viva

Q: What is the difference between a component and a class? **A:** A class is a design-time type; a component is a replaceable, deployable unit that packages several classes behind interfaces. A component is what you build and test as one piece.

Q: What do the lollipop and the socket mean? **A:** The lollipop (circle on a line) is an interface the component provides, that is, implements. The socket (half circle) is an interface the component requires, that is, calls. A socket fits onto a lollipop of the same name.

Q: Why draw interfaces instead of arrows directly between components? **A:** The interface names the contract. The requiring component depends only on the contract, so the providing component can be replaced by any other that provides the same interface.

Q: What is an adapter component? **A:** A component that provides the interface the application wants and requires the interface an external system offers, for example Payment Adapter provides IPayment and requires IGatewayAPI.

Q: What is the difference between a node and an artefact? **A:** A node is a computing resource, a device or an execution environment such as Tomcat. An artefact is a physical file, such as admission.war, that is deployed onto a node.

Q: Why is the web server a separate node from the application server? **A:** It is the only machine that faces the internet. It terminates TLS, serves static files, and forwards to the application server over the internal network, so the application and database servers can be firewalled.

Q: What does the label on a communication path show? **A:** The protocol and usually the port, for example JDBC over TCP 3306. It tells the network team what traffic to allow between the nodes.

Q: How does the deployment diagram relate to the component diagram? **A:** The artefacts on the nodes are the built forms of the components: the web archive holds the application components, the schema is the database component, the static bundle is part of the presentation layer.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Drawing components for individual classes (Student, Question) instead of for cohesive units (Exam Engine, Question Bank).
- Showing a required interface with no component that provides it, or a provided interface that nothing uses.
- Dependency arrows pointing upward, for example the database requiring a service.
- Putting an artefact on a device without the execution environment that runs it, such as admission.war directly on the server with no Tomcat node.
- Leaving communication paths unlabelled; the protocol is the point of the deployment diagram.
- Placing artefacts on client nodes; a browser downloads pages at run time, nothing is deployed there.

Session Summary

■ Write in lab record

- Question 16: component diagram for Online Examination with nine components in Presentation, Application and Data packages and seven named interfaces; PlantUML source `online_examination_components.puml`
- Question 17: component diagram for Order Processing with twelve components including Payment and Shipping adapters and an External package, ten interfaces; PlantUML source `order_processing_components.puml`
- Question 18: deployment diagram for Online Student Admission with client, web, application, database and file server nodes, two external gateways, artefacts and labelled protocols; PlantUML source `admission_deployment.puml`
- Problem description (300 to 500 words) and assumptions for every diagram
- Component and interface tables, node and artefact tables, ASCII sketches, ready to redraw in StarUML and export as PNG

[✎ Edit this page](#)

Session 6

Session 8