

Session 5

Modular design of the Railway Reservation System with structure chart, module specifications, and interfaces

This session turns the Level-1 DFD of Session 4 into a design: a structure chart from one main controller down to the eight modules of the Railway Reservation System (RRS), a specification for every module, and exact signatures for the functions that other modules call. Modular design is the step where the system stops being a description and starts being a plan that a programmer can implement one piece at a time. A good split has modules that each do one job (high cohesion) and share as little as possible (low coupling). The same eight modules are estimated in Session 2, drawn as processes in Session 4, and coded in Session 13, so the names here must match those pages.

Objectives

✕ Do not copy. Read for understanding and the viva

- Draw a structure chart for the RRS from a main controller down to sub-functions.
- Write a module specification (purpose, inputs, outputs, callers, callees, data stores) for all eight modules.
- Judge the design using cohesion and coupling, with examples from the RRS.
- Define the interface of the five most important functions as signatures with parameter tables.
- Record the design decisions taken and the alternatives rejected.

Problem Statement

■ Write in lab record

Session 5: Develop a modular design for RRS.

Concept

✕ Do not copy. Read for understanding and the viva

What a module is

A module is a named unit of the program with one responsibility, a defined interface (what goes in, what comes out) and hidden internals. In Python a module is a file or a group of functions; in C it is a `.c` file with a header. The RRS has eight modules, fixed in Session 1, and each one maps to exactly one Level-1 DFD process from Session 4.

Structure chart

A structure chart shows which module calls which. The top box is the controller that starts the program (the menu in Session 13). Boxes below it are the modules; boxes below those are the sub-functions inside each module. An arrow

means “calls”. Data passed along a call is written on the arrow in the module specification table, not on the chart, so the chart stays readable.

Cohesion

Cohesion is how strongly the parts of one module belong together. From worst to best: coincidental, logical, temporal, procedural, communicational, sequential, functional. Aim for functional cohesion: every function in the module contributes to one task. A module named “Utilities” that holds date formatting, PNR generation and report printing is coincidental cohesion and a sign of a bad split.

Coupling

Coupling is how much one module depends on the inside of another. From best to worst: data, stamp, control, common (shared global data), content (one module edits another’s code or data directly). Aim for data coupling: modules pass only the values they need as parameters and get results back as return values. Common coupling through a global variable is the mistake to watch for in a small Python program, because it is the easiest thing to write.

Design decisions

A design is a set of choices. Write each one down with the alternatives you rejected and the reason. In the viva you are asked why, not what.

Structure Chart

■ Write in lab record

```

1          +-----+
2          |   RRS Main Controller   |
3          | (menu, role check)     |
4          +-----+
5          |
6  +-----+-----+-----+-----+-----+-----+-----+
7  |           |           |           |           |           |           |
8  v           v           v           v           v           v           v
9  +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+
10 | 1   | | 2   | | 3   | | 4   | | 5   | | 6   | | 7   | | 8   |
11 | User | | Train | | Search | | Book- | | Cancel | | Pay- | | Noti- | | Re- |
12 | Mgmt | | and   | | and   | | ing   | | and   | | ment | | fica- | | ports |
13 |      | | Sched. | | Avail. | |      | | Refund | |      | | tion  | |      |
14 +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+
15 |           |           |           |           |           |           |
16 | register | add_train | search_  | validate | lookup_  | initiate | send_    | occupancy
17 | login    | add_route | trains   | _request | pnr        | _payment | booking_ | _report
18 | logout   | add_      | check_   | allocate | compute_   | confirm_ | confirm  | revenue_
19 | get_role | schedule | avail-  | _seats  | refund     | payment  | send_    | report
20 |          | set_fare | ability | generate | promote_   | fail_    | cancel-  | cancel-
21 |          | cancel_  | compute_ | _pnr    | waitlist   | payment  | lation   | lation_
22 |          | schedule | fare    | save_   | record_    | issue_   | send_    | report
23 |          |         |         | booking | refund     | refund   | schedule |
24 |          |         |         |         |         |         | _change  |
25 +-----+-----+-----+-----+-----+-----+-----+
26
27 Cross-module calls (arrow = "calls"):
28 4 Booking      ---> 3 check_availability, 3 compute_fare
29 4 Booking      ---> 6 initiate_payment, 6 confirm_payment
30 4 Booking      ---> 7 send_booking_confirm
31 5 Cancellation ---> 6 issue_refund
32 5 Cancellation ---> 7 send_cancellation
33 2 Schedule     ---> 7 send_schedule_change
34 Main          ---> 1 login (before any other module)

```

Level	Box	Meaning
0	RRS Main Controller	Shows the menu for the logged-in role and dispatches to one module per menu choice
1	Modules 1 to 8	The eight RRS modules from Session 1, one per Level-1 DFD process
2	Sub-functions	Functions inside each module; each becomes one Python function in Session 13
Arrow	Downward line	"Calls"; the caller waits for a return value
Cross-module list	Text under the chart	Calls that cross module boundaries; every one is data coupling (see below)

Module Specifications

■ Write in lab record

Module	Purpose	Inputs	Outputs	Called by	Calls	Data stores touched
1 User Management	Register users, verify login, return the role for the menu	name, email, mobile, password, role	user_id, session token, role	Main Controller	none	User (read, write)
2 Train and Schedule Management	Admin adds trains, routes, stations, coaches, fares and daily schedules; cancels a schedule	train_no, name, type, coaches, station list with times, run_date, rate_per_km	confirmation, schedule_id	Main Controller (Administrator only)	7 Notification (schedule change)	Train, Route, Station, Coach, Seat, Schedule, Fare (write)
3 Search and Availability	Find schedules between two stations on a date; count free seats per class; compute fare	from_station, to_station, run_date, schedule_id, class	list of schedules, available seat count, fare amount	Main Controller, 4 Booking	none	Schedule, Route, Coach, Seat, Booking, Fare (read)
4 Booking	Validate the request, allocate seats or waitlist, generate PNR, save booking and passengers	schedule_id, class, passenger list (max 6), user_id	pnr, status (CONFIRMED or WAITLISTED), total_fare	Main Controller (Passenger, Reservation Clerk)	3 Search and Availability, 6 Payment, 7 Notification	Booking, Passenger, Seat (write), Schedule (read)
5 Cancellation and Refund	Look up PNR, apply the refund rule by hours	pnr, user_id, reason	refund amount, new booking status,	Main Controller (Passenger, Reservation Clerk)	6 Payment, 7 Notification	Booking, Refund, Seat (write), Schedule, Payment (read)

Module	Purpose	Inputs	Outputs	Called by	Calls	Data stores touched
	before departure, free seats, promote waitlist		promoted PNRs			
6 Payment	Start a payment with the gateway, record success or failure, issue refund transactions	pnr, amount, mode	payment_id, status, txn_ref	4 Booking, 5 Cancellation	Payment Gateway (external)	Payment (write)
7 Notification	Send SMS and email for booking, cancellation and schedule change	pnr or schedule_id, event type, recipient mobile and email	delivery status	2 Schedule, 4 Booking, 5 Cancellation	Notification Service (external)	Booking, User (read)
8 Reports	Produce occupancy, revenue and cancellation reports for a date range	from_date, to_date, train_no (optional)	report table	Main Controller (Administrator only)	none	Booking, Payment, Refund, Schedule, Coach (read)

Cohesion and Coupling in this Design

■ Write in lab record

Cohesion

Module	Cohesion type	Why
4 Booking	Functional	Every sub-function (validate, allocate seats, generate PNR, save) is a step of one task: create a booking
5 Cancellation and Refund	Functional	Lookup, refund calculation, seat release and waitlist promotion all serve one task: cancel a PNR
6 Payment	Functional	Initiate, confirm, fail and refund all deal with one payment record
3 Search and Availability	Sequential	Output of <code>search_trains</code> (<code>schedule_id</code>) is the input of <code>check_availability</code> , whose output feeds <code>compute_fare</code>
2 Train and Schedule Management	Communicational	Add train, add route, add schedule and set fare all work on the same train master data, but are separate admin tasks
7 Notification	Logical	Three sends chosen by event type share one channel; acceptable because the module is thin and calls an external service
1 User Management	Functional	Register, login, logout and <code>get_role</code> all maintain one thing: who the current user is
8 Reports	Communicational	Three reports read the same booking and payment data; each report is independent

The two modules a marker looks at hardest, Booking and Cancellation, are functionally cohesive. Notification is the weakest at logical cohesion, and it is kept small so that this does not matter.

Coupling

Data coupling, the best kind, occurs at every cross-module call:

- Booking calls `check_availability(schedule_id, travel_class)` and receives an integer. Booking does not read the Seat table itself.
- Booking calls `initiate_payment(pnr, amount, mode)` and receives a `payment_id` and status. Booking never sees the gateway response.
- Cancellation calls `send_cancellation(pnr)` and receives a delivery flag. Notification looks up the mobile and email itself.

Stamp coupling occurs once: Booking passes the whole passenger list (a list of dictionaries with name, age, gender, berth_preference) to `allocate_seats`. This is accepted because the function needs every field to match berth preference to berth type.

Control coupling was removed: an early draft had `send_notification(pnr, kind)` where `kind` was a flag that chose the message. It is replaced by three functions (`send_booking_confirm`, `send_cancellation`, `send_schedule_change`) so the caller does not steer the callee's logic.

Common coupling was avoided in one specific place. The obvious Python shortcut is a global `current_booking` dictionary that Booking fills, Payment reads, and Notification reads again. Instead, `book_ticket` returns the `pnr`, and Payment and Notification each receive the `pnr` as a parameter and read the Booking store themselves. The only shared state is the persistent data stores, accessed through named load and save functions, never through module-level variables.

Interface Definitions

■ Write in lab record

The five functions below are the ones other modules or the main controller call. All parameters are passed by value; all results are return values. Errors are raised as exceptions named in the last row of each table.

search_trains

`search_trains(from_station, to_station, run_date)` returns `list of schedule`

Parameter	Type	Constraint
from_station	string, station_code	Must exist in Station
to_station	string, station_code	Must exist in Station, not equal to from_station
run_date	date, YYYY-MM-DD	Today or up to 120 days ahead
returns	list of (schedule_id, train_no, name, departure_time, arrival_time, distance_km)	Empty list if no train; never an error
raises	UnknownStationError	If either code is not in Station

check_availability

`check_availability(schedule_id, travel_class)` returns `available_count`

Parameter	Type	Constraint
schedule_id	integer	Must exist in Schedule with status not CANCELLED
travel_class	string, one of SL, 3A, 2A, 1A	Must have a Coach of this class on the train
returns	integer	Seats in class minus CONFIRMED bookings; 0 means next booking is WAITLISTED
raises	ScheduleNotFoundError, ClassNotOnTrainError	As named

book_ticket

`book_ticket(schedule_id, travel_class, passengers, user_id)` returns `pnr`

Parameter	Type	Constraint
<code>schedule_id</code>	integer	Must exist; departure_time must be in the future (a schedule cannot be booked after departure)
<code>travel_class</code>	string, one of SL, 3A, 2A, 1A	As in check_availability
<code>passengers</code>	list of (name, age, gender, berth_preference)	1 to 6 entries; age 1 to 120; gender M, F or O; berth_preference LOWER, MIDDLE, UPPER, SIDE_LOWER, SIDE_UPPER or NONE
<code>user_id</code>	integer	Logged-in Passenger or Reservation Clerk
<code>returns</code>	string, 10-digit pnr	Unique; booking saved with status CONFIRMED or WAITLISTED and total_fare filled
<code>raises</code>	DepartedScheduleError, TooManyPassengersError, PaymentFailedError	Payment failure leaves no booking record

initiate_payment

`initiate_payment(pnr, amount, mode)` returns `(payment_id, status, txn_ref)`

Parameter	Type	Constraint
<code>pnr</code>	string, 10 digits	Must exist in Booking
<code>amount</code>	decimal, rupees	Greater than 0, equal to Booking.total_fare
<code>mode</code>	string, one of UPI, CARD, NETBANKING, CASH	CASH allowed only for Reservation Clerk
<code>returns</code>	payment_id integer, status SUCCESS or FAILED, txn_ref string from gateway	A FAILED status is a normal return, not an exception
<code>raises</code>	GatewayUnreachableError	Only when the gateway does not answer

cancel_ticket

`cancel_ticket(pnr, user_id, reason)` returns `refund_amount`

Parameter	Type	Constraint
pnr	string, 10 digits	Must exist; status CONFIRMED or WAITLISTED; not already CANCELLED
user_id	integer	Must be the booking owner or a Reservation Clerk
reason	string, up to 100 characters	Stored in Refund.reason; may be empty
returns	decimal, rupees	100% minus flat clerkage if more than 48 hours before departure; 50% between 48 and 12 hours; 0 under 12 hours
raises	PnrNotFoundError, AlreadyCancelledError, NotOwnerError	Waitlist promotion happens before return

Design Decisions

■ Write in lab record

Decision	Alternatives	Chosen	Reason
Where seat allocation lives	Inside Search and Availability; inside Booking; a separate Seat module	Inside Booking	Allocation changes Seat and Booking together in one transaction; splitting it would need two modules to agree on a lock
How Payment learns the booking amount	Global variable; Booking passes amount; Payment reads Booking by pnr	Booking passes pnr and amount	Data coupling; Payment can also verify amount against Booking.total_fare
Notification interface	One function with a kind flag; one function per event	One function per event	Removes control coupling; each function can format its own message
Refund rule location	In Payment; in Cancellation and Refund; in a shared rules module	In Cancellation and Refund	The rule is about time before departure, which Cancellation already computes; Payment only moves money
Waitlist promotion trigger	Scheduled batch job; on every cancellation	On every cancellation	Simpler, immediate, no scheduler needed for a lab-scale system
Fare computation	Stored per booking only; computed from Fare.rate_per_km times Route distance	Computed by <code>compute_fare</code> in Search and Availability, then stored in Booking.total_fare	Search must show fare before booking; storing the result keeps the historical fare after a rate change
Persistence	Relational database; JSON file; in-memory only	JSON file (<code>rrs_data.json</code>) behind load and save functions	Session 13 forbids third-party packages; a database can replace the file later without touching module logic
Role check location	In every module; in the main controller only	In the main controller, with a second check in admin-only functions	Menu never shows an option the role cannot use; the second check protects against direct calls

Viva Questions

× Do not copy. Read for understanding and the viva

- **Q:** What is the difference between a structure chart and a DFD? **A:** A DFD shows data movement between processes; a structure chart shows which module calls which and is a design artefact, not an analysis one.

- **Q:** Which RRS module has the highest cohesion and why? **A:** Booking: every sub-function is a step in creating one booking, which is functional cohesion.
- **Q:** Give one example of data coupling in the RRS. **A:** Booking calls `check_availability(schedule_id, travel_class)` and receives only an integer.
- **Q:** Where could common coupling have crept in, and how was it avoided? **A:** A global `current_booking` dictionary shared by Booking, Payment and Notification; avoided by returning the pnr and passing it as a parameter.
- **Q:** Why is the refund rule not in the Payment module? **A:** The rule depends on hours before departure, which Cancellation computes; Payment only records money movement.
- **Q:** Why does `book_ticket` reject a schedule after departure? **A:** Business rule: a schedule cannot be booked after departure; the check is at the entry to Booking so no later step sees a departed schedule.
- **Q:** What kind of coupling is passing the whole passenger list to `allocate_seats`? **A:** Stamp coupling; accepted because every field is used.
- **Q:** What does the main controller do besides show a menu? **A:** It calls login first and dispatches only the menu options the returned role is allowed to use.
- **Q:** How many modules does the design have, and where else do the same eight appear? **A:** Eight; in the Session 2 function point count, the Session 4 Level-1 DFD, and the Session 13 program.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Drawing the structure chart with data flows on the arrows, which turns it back into a DFD. Data goes in the module specification table.
- Inventing a ninth "Database" or "Utility" module. Data stores are not modules; they appear in the "Data stores touched" column.
- Using module names that differ from Sessions 1, 2 and 4. The examiner cross-checks; use the same eight names.
- Writing signatures without types and constraints. `book_ticket(a, b, c, d)` earns nothing; the parameter table is the deliverable.
- Claiming "low coupling" without an example. Name the call, the parameters, and the return value.
- Leaving the design decision table with a chosen column but no alternatives. A decision with no alternative is not a decision.

Session Summary

■ Write in lab record

- Structure chart with the main controller, eight modules, and sub-functions, plus the box and arrow legend table.
- Module specification table with all eight rows filled.
- Cohesion table and coupling notes with at least three named examples, including the avoided global variable.
- Five interface definitions with parameter tables.
- Design decision table with at least six rows.

✎ Edit this page

< Previous
Session 4

Next >
Session 6