Design Document for <<ComScheduler>>

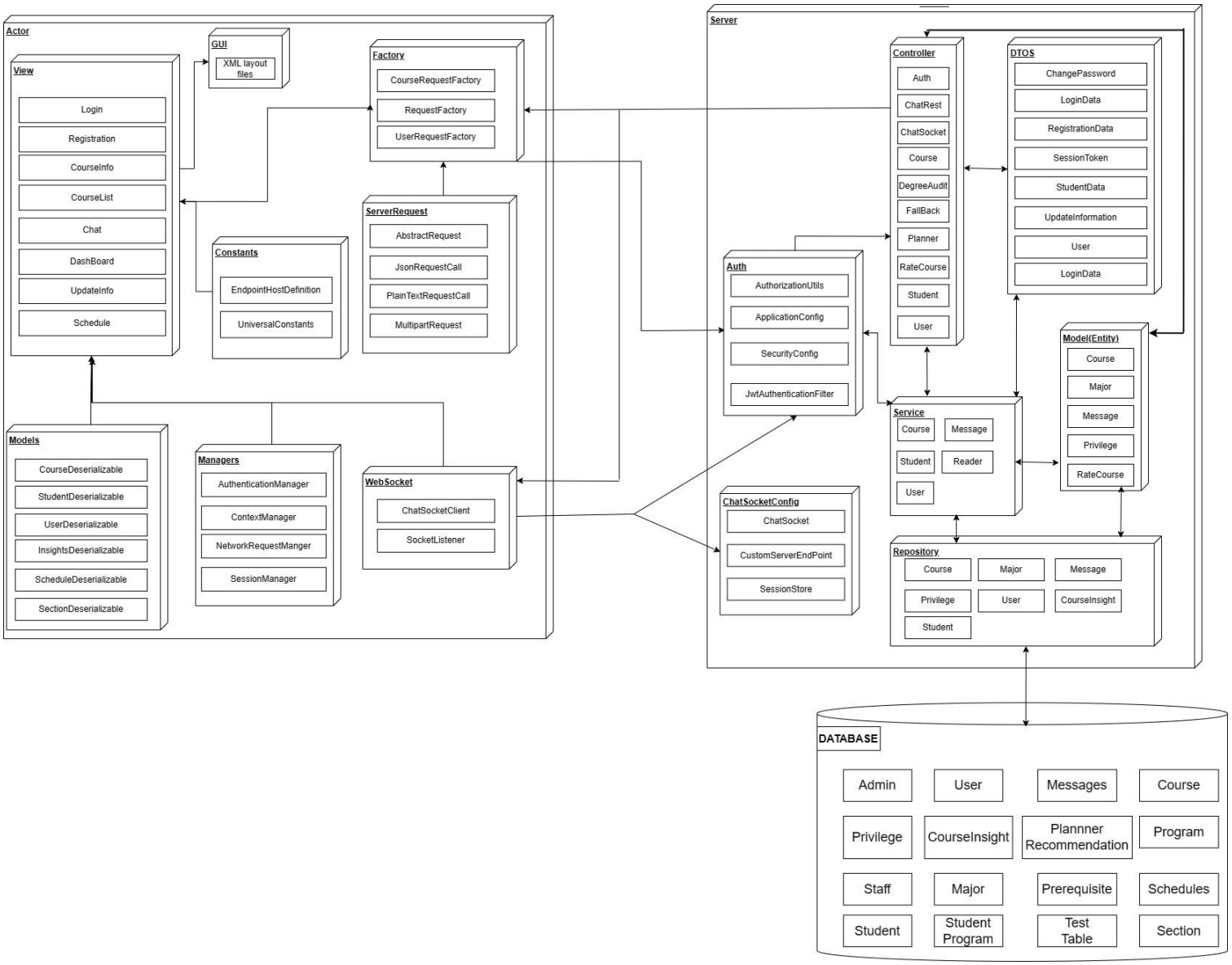
Group <lg-112>

Member1 Name: Khoi Pham 25% contribution(block diagram & backend description)

Member2 Name: Nhat Anh Bui 25% contribution(block diagram)

Member3 Name: Thanh Mai 25% contribution (design description)

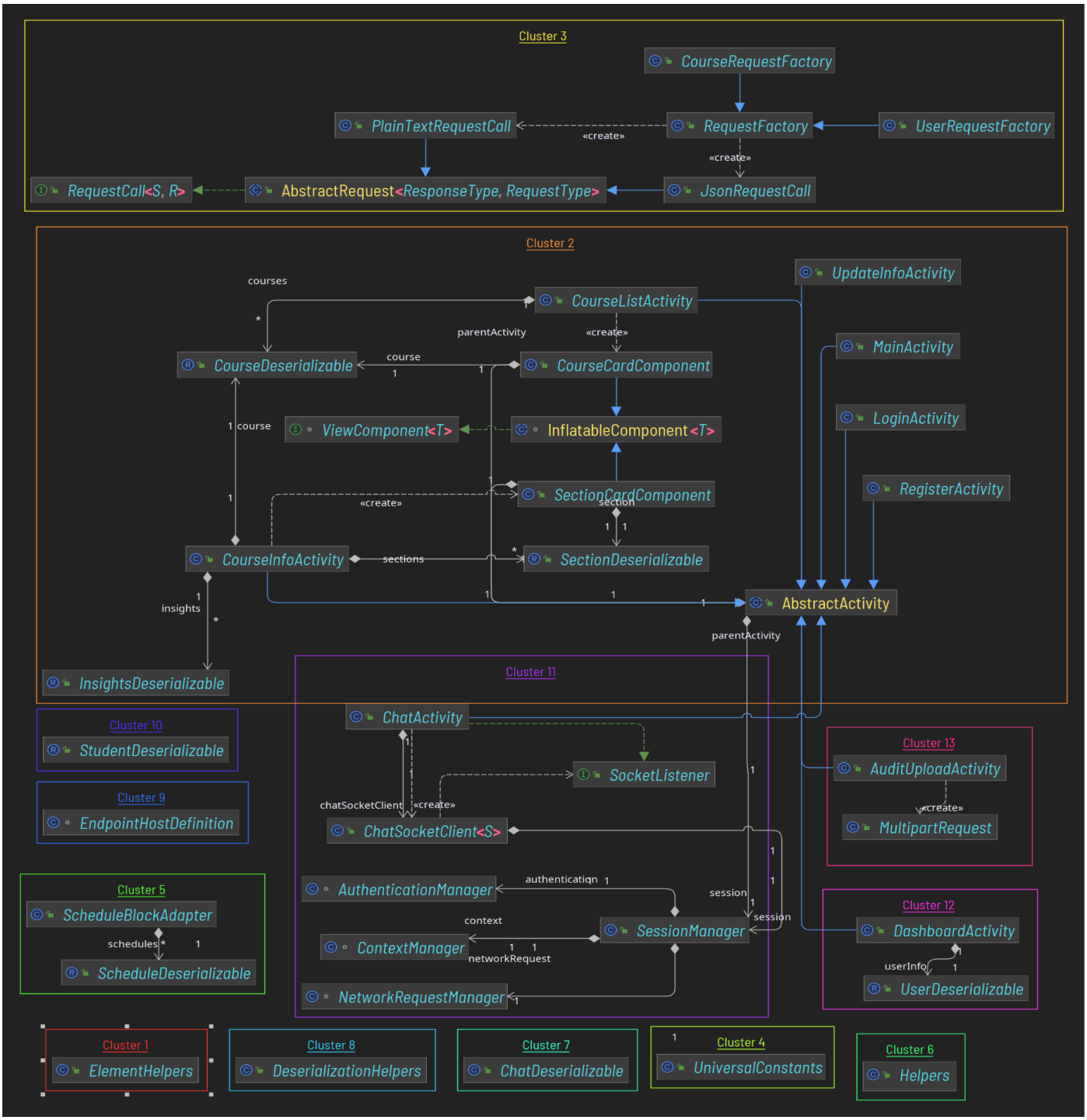
Member4 Name: Kim Sang Huynh 25% contribution(block diagram)


Actor
GUI
XML layout files
View
Login
Registration
CourseInfo
CourseList
Chat
DashBoard
UpdateInfo
Schedule
Factory
CourseRequestFactory
RequestFactory
UserRequestFactory
ServerRequest
AbstractRequest
JsonRequestCall
PlainTextRequestCall
MultipartRequest
Constants
EndpointHostDefinition
UniversalConstants
Models
CourseDeserializable
StudentDeserializable
UserDeserializable
InsightsDeserializable
ScheduleDeserializable
SectionDeserializable
Managers
AuthenticationManager
ContextManager
NetworkRequestManger
SessionManager
WebSocket
ChatSocketClient
SocketListener
Server
Controller
Auth
ChatRest
ChatSocket
Course
DegreeAudit
FallBack
Planner
RateCourse
Student
User
DTOS
ChangePassword
LoginData
RegistrationData
SessionToken
StudentData
UpdateInformation
User
LoginData
Auth
AuthorizationUtils
ApplicationConfig
SecurityConfig
JwtAuthenticationFilter
Model(Entity)
Course
Major
Message
Privilege
RateCourse
Service
Course
Message
Student
Reader
User
ChatSocketConfig
ChatSocket
CustomServerEndPoint
SessionStore
Repository
Course
Major
Message
Privilege
User
CourseInsight
Student
DATABASE
Admin
User
Messages
Course
Privilege
CourseInsight
Plannner Recommendation
Program
Staff
Major
Prerequisite
Schedules
Student
Student Program
Test Table
Section

(I know that this section is supposed to be one page, but most of the occupied space are the images and the text can easily fit in one page)

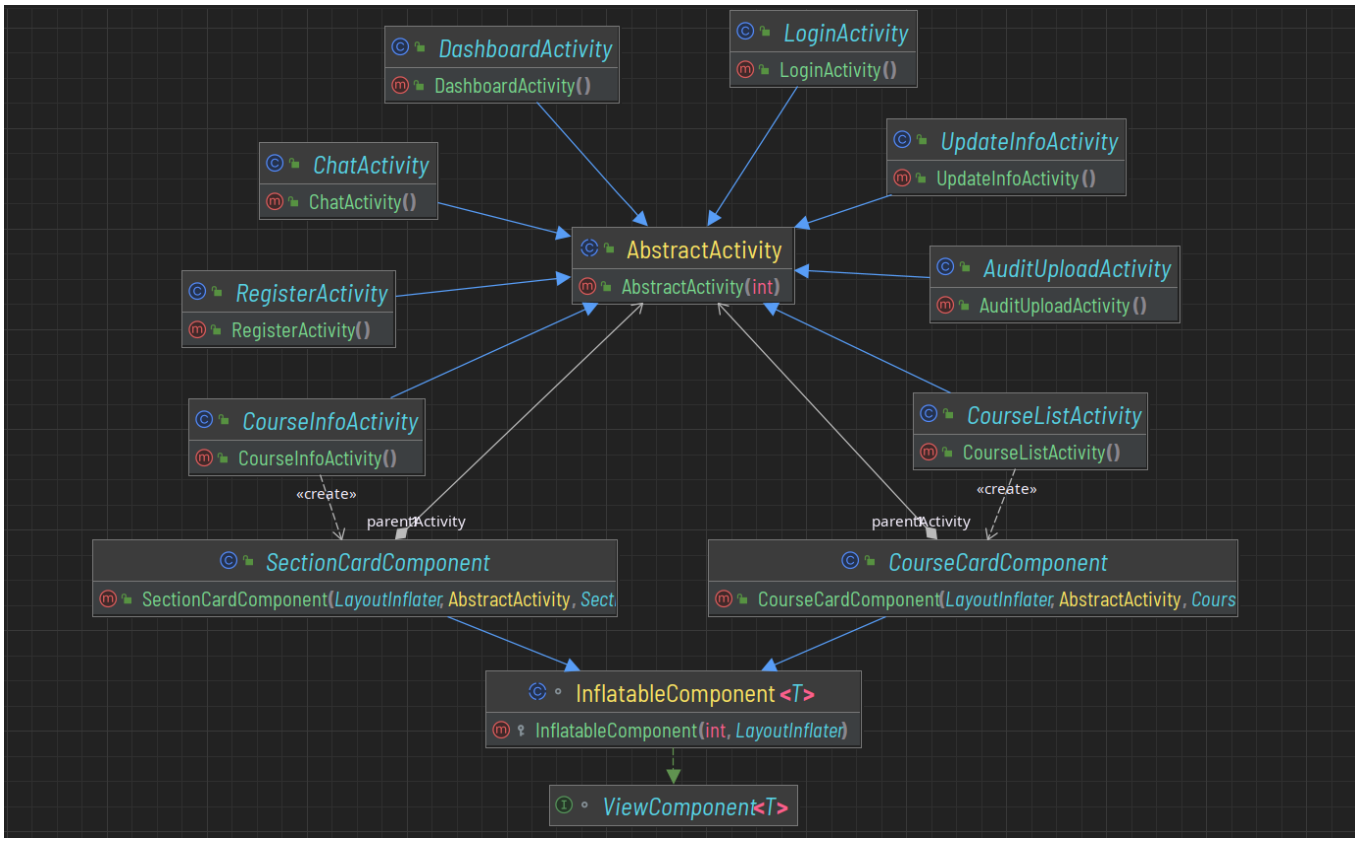
Frontend - underlying hidden architecture:

This section elaborates more on the hidden architecture in the frontend that is not reflected in the main diagram for the sake of brevity and readability. The complete picture of the frontend is given below (explanations follow)



- Activities and components:

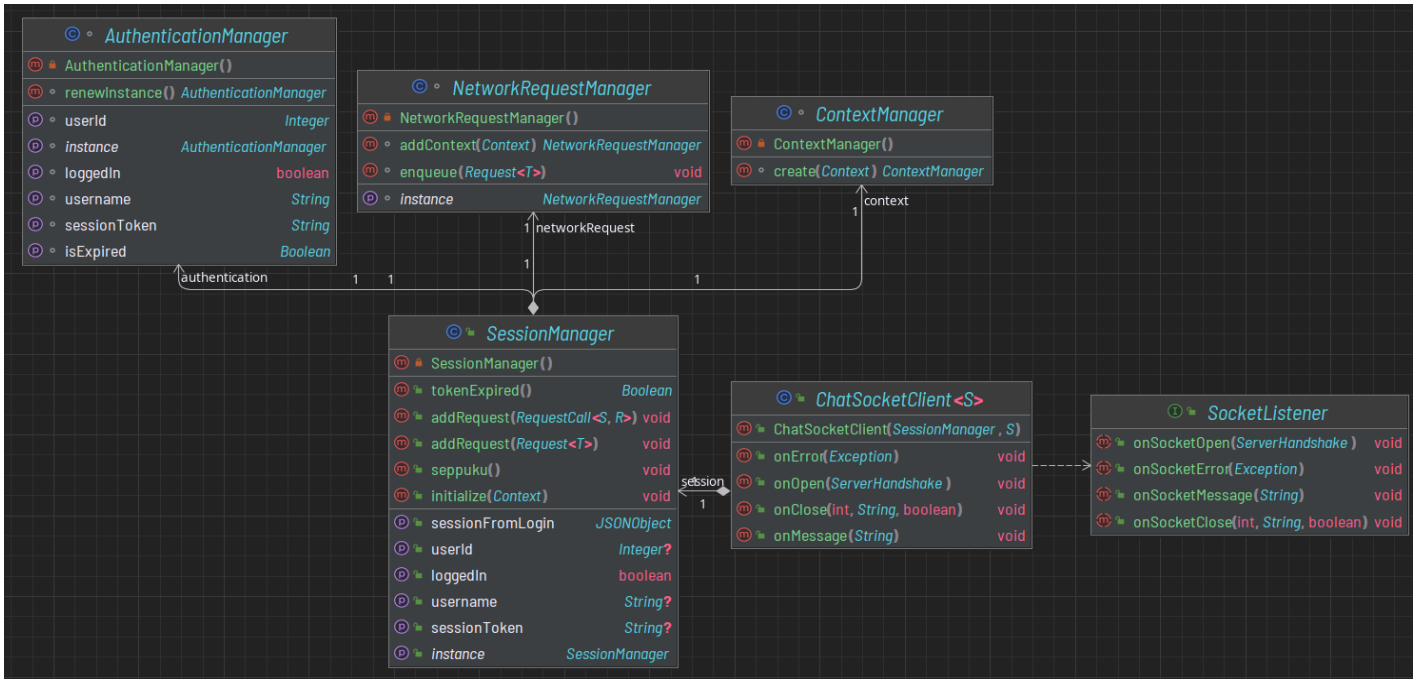
Each activity inherits the parent “AbstractActivity” class, which provides a unified implementation for various functionalities that are frequently used within activities. A component is any layout that is meant to be inflated during runtime (e.g. course entries, section entries), the “InflatableComponent” class enforces a common procedure to build and inflate these components.



Activities and components' object hierarchy and dependency graph

- Manager singletons:

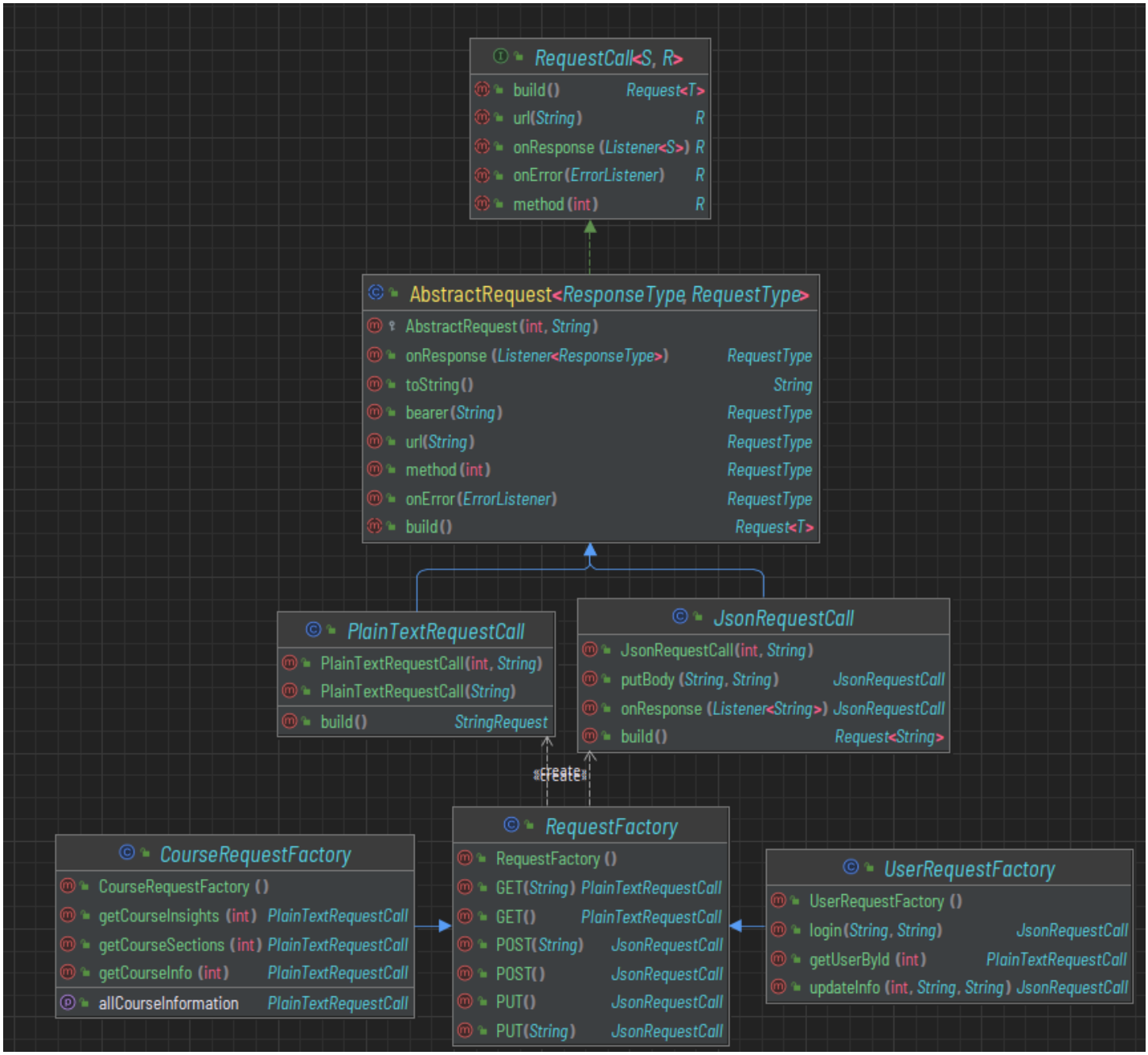
Managers are singletons that hold states that are crucial to operations throughout various aspects of the application (for example, the current session, user information, the request queue, etc.). Of the handful of singletons each designated to one set of tasks, only "SessionManager" is public and exposed to other aspects of the application, whereas the other classes are package-private. As it is injected into "AbstractActivity", every activity can access and interact with various states of the application.



Various singletons and their dependencies

- Requests and request factories:

Requests in the frontend application are wrapped with a builder pattern to mitigate developer mistakes and improve maintainability. It also provides a nicer programming interface compared to Volley's. Request factories take in a set of predefined parameters and return a fully filled out request. Additionally, it handles authentication during API calls, and also tries to ping every possible backend host address upon startup to try to guess the host that is being used at the moment.



Requests and their factories' object hierarchy and dependency graph

- Deserializables:

These are just data classes (Java records) that have extra attached factory methods that take in a JSONObject and return the corresponding data class. This is a more elegant way to deserialize JSON strings, with a cleaner

programming interface to carry out routine deserialization operations such as type validation, data validation, nullish coalescence, array deserialization, etc.

Backend - JWT authorization

The backend enforces an authorization system based on JSON web tokens. A JWT token is generated with all the necessary claims whenever a user logs in. All future requests to the backend server must also include this token in order for the request to be served. The token includes a claim for the user ID, allowing the backend to serve the correct data and validate whether the user has sufficient privileges to carry out certain actions.

Before the requests reach the Controller, it has to pass the JWT Authorization Filter, which will take the token from the request header “Authorization”, decode the token, extract the claim username, and then check by calling the UserService to see whether there is that username in the database or not. If satisfied, it will let the request reach the Controller. Otherwise, it will return 403 Forbidden in the status.

Database Schema