UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED ELETTRICA E MATEMATICA APPLICATA

Report - Software Architectures for Enterprise Systems

Peer Review System: PeerFlow
Architecture Document
Group 1

Professor

Nicola Capuano - ncapuano@unisa.it

Team members

Name and Surname	Student ID	E-mail
Ciaravola Giosuè	0622702177	g.ciaravola3@studenti.unisa.it
Della Corte Mario	0622702354	m.dellacorte19@studenti.unisa.it
Polverino Alessandro	0622702352	a.polverino15@studenti.unisa.it

ACADEMIC YEAR 2024/2025

Contents

1 Module Views

1.1 Global Module View

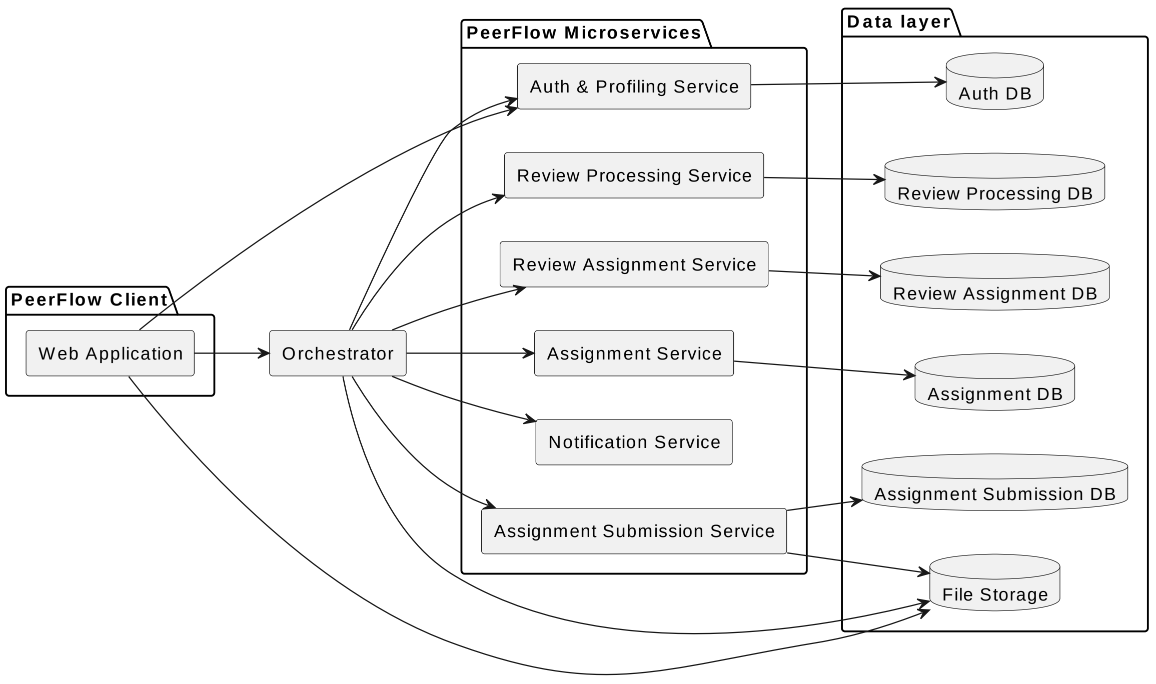

Figure 1.1: Global module view of PeerFlow.

1.1.1 Elements

1. **Web Application**

   This component serves as a monolithic front-end application that Students and Teachers may use to access the system's services. This component directly interacts with the Auth & Profiling Service for authentication purposes; for all the other functionalities, this component interfaces with the Orchestrator service. Furhermore, it can ask for files directly to the FileStorage, which exposes a read-public HTTP API.

2. **Auth & Profiling Service**

   This component exposes a REST API that allows users to sign up, log in, create, and refresh JWT tokens, and check the existence of users.

3. **Assignment Service**

   This component exposes a REST API that allows Teachers to create assignments within the course (specifying Students) and to edit them.

4. **Assignment Submission Service**

   This component exposes a REST API that allows Students to submit their work (mandatory text and optional files) for the assignment.

5. **Review Assignment Service**

   This component exposes a REST API that allows Teachers to issue a Peer Review assignment for the students in the assignment (either manually or automatically specifying pairings).

6. **Review Processing Service**

   This component exposes a REST API that allows Students to submit their reviews for the assignments they have been assigned to.

7. **Notification Service**

   This service exposes a REST API that allows to send email notification to the users of the system.

8. **Orchestrator**

   This service exposes a REST API that the front-end application uses to interact with the system's services. It acts as an intermediator capable of integrating more services, also exposing them to the final user as a single web endpoint.

9. **File Storage Service**

   This service exposes a REST API that manages files in an efficient and scalable way.

Each service (elements 1-7) has its own database istance, which is compliant with the microservice architecture. Data structures are described in the dedicated section.

1.1.2 Rationale

The architectural design for PeerFlow, centered around a Microservices Architecture orchestrated by a dedicated orchestrator, addresses the core requirements and challenges of building a scalable, maintainable, and robust peer review system for MOOC environments.

Microservices Architecture

The adoption of a Service-Oriented/Microservices Architecture is a direct response to several key requirements and assumptions:

- **Scalability and High Availability:** PeerFlow "will be designed to support a large and highly variable number of users, typical of MOOC environments like Coursera or Udacity" and its architecture "will be based on a Service Oriented/Microservices Architecture approach, to ensure horizontal scalability, high availability, and low latency." Microservices support horizontal scalability by allowing individual services to be scaled independently based on their load. For instance, if there is a peak in assignment submissions, only the Assignment Submission Service needs to scale up, not the entire application. Similarly, high availability is achieved because the failure of one service (e.g., Review Processing Service as per QA-AV-1 scenario) does not necessarily bring down the entire system, thanks to isolation and independent deployment.

- **Modifiability and Maintainability:** The non-functional requirements emphasize modifiability. For example, adding a new mandatory field to the student data model (QA-MO-01) or a new submission file type (QA-MO-2) is expected to be completed within a short timeframe with localized changes. A microservices approach ensures that changes have to be made to specific services, reducing the risk of introducing bugs in unrelated parts of the system and allowing for faster development cycles.

- **Technology Heterogeneity (Implied):** While not explicitly stated as a requirement, microservices allow for different technologies to be used for different services, optimizing each service for its specific function. This flexibility enhances future adaptability and leverages specialized tools.

- **Independent Deployment:** The Deployability non-functional requirements, such as deploying a bug fix to a specific service (QA-DE-1) or rolling back a malfunctioning service (QA-DE-2), are directly supported by the independent deployability of microservices. This minimizes downtime and risk during updates.

Dedicated Microservices

Each service in the proposed architecture (Auth & Profiling Service, Assignment Service, Assignment Submission Service, Review Assignment Service, Review Processing Service, Notification Service, File Storage Service) corresponds to a distinct functional area of the PeerFlow system, aligning with the "single responsibility principle" of microservices.

- **Auth & Profiling Service:** Handles user authentication (sign-up, login, JWT tokens) and user profile management, directly addressing FR-SYS-001 (two distinct user roles), FR-SYS-002 (user authentication), and FR-SYS-003 (new user sign-up). Its isolation ensures security concerns are centralized and managed effectively.

- **Assignment Service:** Manages the creation, modification, and viewing of assignments by teachers, addressing FR-ASG-001, FR-ASG-002, FR-ASG-003, FR-ASG-004. It also handles the listing of assignments for students (FR-ASG-006).

- **Assignment Submission Service:** Dedicated to handling student submissions, including text and file attachments (FR-SUB-001, FR-SUB-002, FR-SUB-003). Its separate nature allows for robust handling of potentially large file uploads and associated storage.

- **Review Assignment Service:** Manages the definition of rubrics (FR-PRV-001, FR-PRV-002, FR-PRV-003) and the initiation of the peer review process, including automatic or manual peer assignment (FR-PRV-004). This separation ensures that the complex logic of peer assignment is isolated.

- **Review Processing Service:** Focuses on the actual review process by students, including viewing assigned submissions and submitting reviews (FR-REV-001, FR-REV-002). This separation allows for efficient processing of review data.

- **Notification Service:** Handles sending email notifications to users (FR-ASG-005, FR-PRV-005). This service is designed for integrability, as indicated by scenario QA-IN-1.

- **File Storage Service:** Its inclusion is crucial for managing file attachments, as students must be able to submit file attachments (`FR-SUB-001`), view or download their own submitted work (`FR-SUB-002`), and modify their submitted work (`FR-SUB-003`). It provides a scalable and efficient way to handle binary data separately from relational databases.

Orchestrator Service (API Gateway)

The Orchestrator acts as an API Gateway, an essential pattern in microservices architectures:

- **Simplifies Client-Side Development:** The Web Application interacts with a single Orchestrator endpoint for most functionalities, rather than needing to know and manage multiple microservice endpoints. This simplifies the front-end logic and reduces coupling between the UI and individual services.
- **Request Routing and Composition:** The Orchestrator can route requests to the appropriate microservices and potentially compose responses from multiple services before sending them back to the client. This is crucial for functionalities that span multiple services (e.g., displaying assignment details that might involve data from Assignment Service and Assignment Submission Service).
- **Cross-Cutting Concerns:** The Orchestrator is an ideal place to handle cross-cutting concerns such as load balancing, caching, request logging, and potentially security (though authentication is handled by Auth & Profiling Service).
- **Enables Evolution:** Changes to backend microservices can be abstracted away from the client by modifying the Orchestrator's routing logic, ensuring backward compatibility for the Web Application.
- **Reduces Load on Assignment Submission Service during high peak times:** The ability of the Orchestrator to directly interact with the File Storage Service allows to lighten the load on the Assignment Submission Service and increase its throughput, therefore lowering the need to scale this service. This is compliant with the `QA-PE-3` requirement.

Dedicated Databases per Service

Each microservice having its own database instance (Auth DB, Assignment DB, Review Assignment DB, Review Processing DB, Assignment Submission DB) is a crucial point of the microservices paradigm:

- **Loose Coupling and Data Independence:** This prevents services from being tightly coupled through a shared database, which is a common monolith anti-pattern. Each service can evolve its data schema independently without impacting others.
- **Data Consistency and Autonomy:** Each service is responsible for its own data, ensuring data consistency within its boundaries. This also allows different services to use different types of databases if optimal for their specific data needs.
- **Scalability:** Databases can be scaled independently along with their associated services, as seen in `QA-AV-2` (Scheduled Database Maintenance) where read operations can switch to a replica, and `QA-PE-3` (Submission Peak) where underlying storage infrastructure scales.

Web Application (Client)

The Web Application serves as the user interface, providing accessibility via a web platform as required. Its interaction model is consistent with modern web development practices:

- **Direct Authentication Interaction:** The WebApp directly interacts with the Auth & Profiling Service for authentication. This is a common and secure pattern, as authentication is a critical first step.
- **Direct File Storage Interaction:** The WebApp directly interacts with the File Storage Service for viewing and downloading submitted files directly on the UI. This is a common pattern, as file visualization is a function specific to the UI and it should not encumber the Orchestrator.
- **Orchestrator for Other Functionalities:** For all other functionalities, the WebApp interacts with the Orchestrator. This simplifies client-side development, leverages the Orchestrator's routing capabilities, and allows the backend architecture to evolve more independently of the frontend.

In summary, the chosen architecture effectively leverages the benefits of microservices — scalability, modifiability, independent deployment, and resilience—to meet the complex and evolving demands of a peer review system for large-scale online learning environments. The Orchestrator acts as a crucial unifying layer, simplifying client interactions and managing the distributed nature of the system.

1.2 Data View

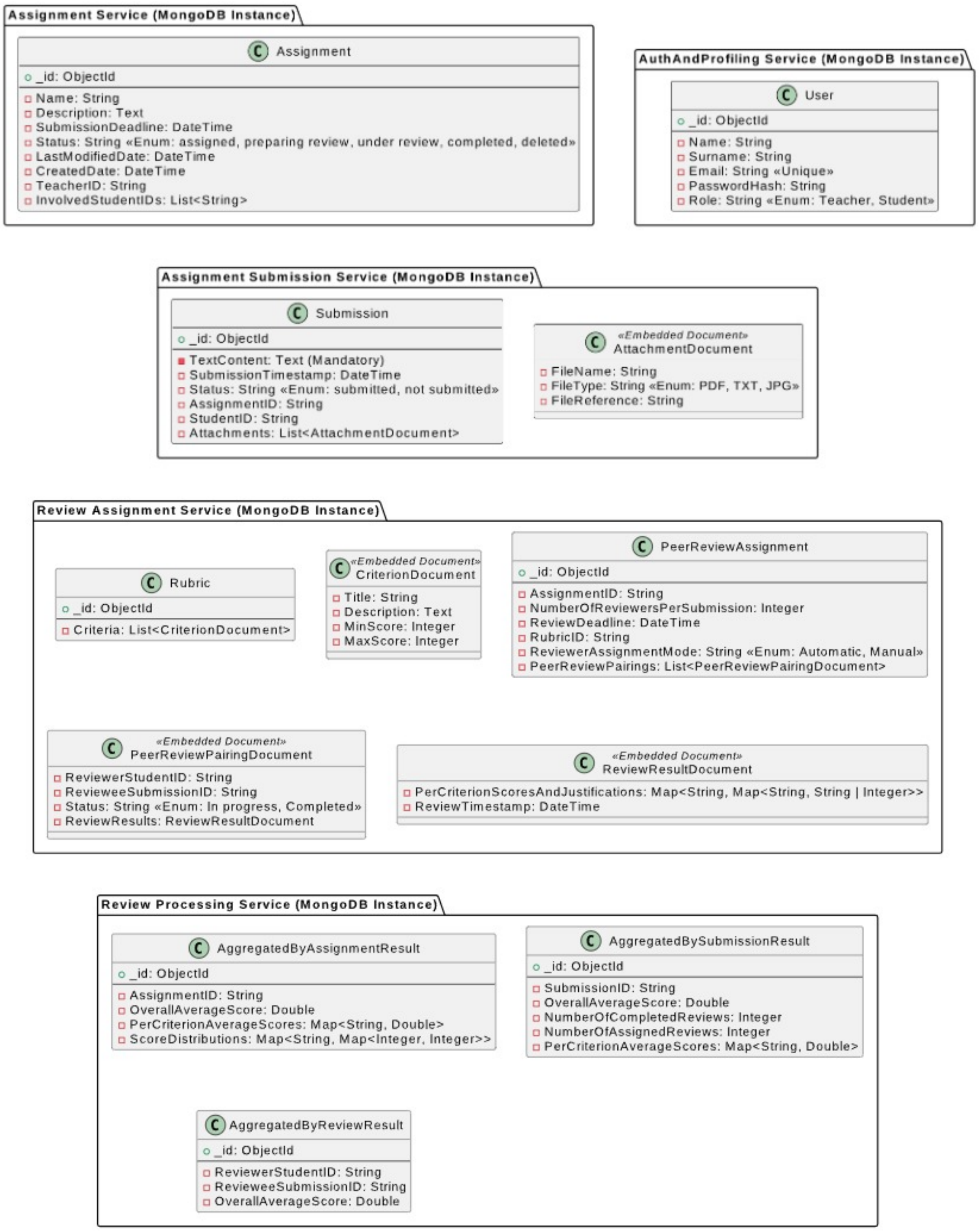
Assignment Service (MongoDB Instance)
Assignment
_id: ObjectId
Name: String
Description: Text
SubmissionDeadline: DateTime
Status: String «Enum: assigned, preparing review, under review, completed, deleted»
LastModifiedDate: DateTime
CreatedDate: DateTime
TeacherID: String
InvolvedStudentIDs: List<String>
AuthAndProfiling Service (MongoDB Instance)
User
_id: ObjectId
Name: String
Surname: String
Email: String «Unique»
PasswordHash: String
Role: String «Enum: Teacher, Student»
Assignment Submission Service (MongoDB Instance)
Submission
_id: ObjectId
TextContent: Text (Mandatory)
SubmissionTimestamp: DateTime
Status: String «Enum: submitted, not submitted»
AssignmentID: String
StudentID: String
Attachments: List<AttachmentDocument>
«Embedded Document»
AttachmentDocument
FileName: String
FileType: String «Enum: PDF, TXT, JPG»
FileReference: String
Review Assignment Service (MongoDB Instance)
Rubric
_id: ObjectId
Criteria: List<CriterionDocument>
«Embedded Document»
CriterionDocument
Title: String
Description: Text
MinScore: Integer
MaxScore: Integer
PeerReviewAssignment
_id: ObjectId
AssignmentID: String
NumberOfReviewersPerSubmission: Integer
ReviewDeadline: DateTime
RubricID: String
ReviewerAssignmentMode: String «Enum: Automatic, Manual»
PeerReviewPairings: List<PeerReviewPairingDocument>
«Embedded Document»
PeerReviewPairingDocument
ReviewerStudentID: String
RevieweeSubmissionID: String
Status: String «Enum: In progress, Completed»
ReviewResults: ReviewResultDocument
«Embedded Document»
ReviewResultDocument
PerCriterionScoresAndJustifications: Map<String, Map<String, String | Integer>>
ReviewTimestamp: DateTime
Review Processing Service (MongoDB Instance)
AggregatedByAssignmentResult
_id: ObjectId
AssignmentID: String
OverallAverageScore: Double
PerCriterionAverageScores: Map<String, Double>
ScoreDistributions: Map<String, Map<Integer, Integer>>
AggregatedBySubmissionResult
_id: ObjectId
SubmissionID: String
OverallAverageScore: Double
NumberOfCompletedReviews: Integer
NumberOfAssignedReviews: Integer
PerCriterionAverageScores: Map<String, Double>
AggregatedByReviewResult
_id: ObjectId
ReviewerStudentID: String
RevieweeSubmissionID: String
OverallAverageScore: Double

Figure 1.2: Data view of the system.

This section details the data view of the system, which is implemented using MongoDB. The data model is designed to leverage MongoDB's document-oriented capabilities, emphasizing data embedding for efficient data retrieval and reduced complexity in data operations. The data view is structured as follows:

- **Collections:** Each top-level entity, represented by a distinct box in the diagram without the `<<embedded document>>` stereotype, corresponds directly to a MongoDB collection.

The fields listed within these entities define the schema for documents stored within their respective collections, specifying both field names and their corresponding data types.

- **Embedded Documents:** Entities explicitly labeled with the `<<embedded document>>` stereotype do not constitute separate MongoDB collections. Instead, they define the structure of sub-documents that are embedded within documents of other collections.

Consider the `PeerReviewAssignment` entity. This entity maps to the `PeerReviewAssignment` MongoDB collection. Documents within this collection will encapsulate details pertinent to a peer review assignment, including fields such as `_id` (a unique identifier), `AssignmentId`, `NumberOfReviewersPerSubmission`, `ReviewDeadline`, `RubricId`, and `ReviewAssignmentMode`.

A key aspect of this document's structure is the `PeerReviewPairings` field. This field is designed as a list of embedded documents. The precise structure of each element within this list is dictated by the `PeerReviewPairingDocument` entity, which is marked as an `<<embedded document>>`. Consequently, each embedded document within the `PeerReviewPairings` list will contain fields defining a specific review pairing, such as `ReviewerStudentID`, `RevieweeSubmissionID`, `Status` (indicating the progress of the review), and `ReviewResults` (another embedded document detailing the review outcomes).

1.2.1 Rationale

The embedded document strategy is employed to maintain data locality, optimize query performance by reducing the need for explicit joins, and align with MongoDB's best practices for data modeling. It allows for the retrieval of all relevant pairing information directly within the parent assignment document, enhancing data access efficiency.

2 Component And Connector Views

Given that a comprehensive Components and Connectors (C&C) view of the entire system could prove overly dense and complex for detailed analysis, this chapter will instead present a series of focused views. These views are specifically tailored to illustrate the architectural components and their interconnections pertaining to the system's principal use cases.

2.1 General Details

Some components are repeated in subsequent paragraphs and share common details and functionalities:

1. **Orchestrator:** This service acts as the API Gateway for the PeerFlow system.
    - **IAPIGateway interface - Entrypoint:** It receives HTTP requests from the Web Application and interacts with components of the PeerFlow system to accomplish the requested task.
    - **Authorization Enforcement:** The Orchestrator is capable of verifying the JWT included in the requests it receives by using the public key of the Auth & Profiling Service. It stores the public key in a local cache with a predetermined TTL and refreshes it when the TTL expires.
2. **Auth & Profiling Service:** This microservice is responsible for user authentication and profile management.
    - **IPublicKey interface - Asymmetric Signature verification:** This interface allows other services to request the public key that can be used to verify the signature of the JWTs.
    - **Auth DB:** This is the dedicated database instance for the Auth & Profiling Service. It stores user profiles and roles, accessed for authorization checks and to retrieve user(s) data.
3. **Notification Service:** This service exposes a REST API that allows to send email notifications to the users of the system.
    - **INotification interface:** Receives `sendNotification()` calls from the Orchestrator and dispaches email notifications to selected users.
4. **Other Services:** This aggregated component represents domain-specific microservices within the PeerFlow system that are not specifically relevant in the view.

- **IOtherServicesAPI:** This interface is an abstraction and aggregates all the interface provided by the Other Services of the system, shown for completeness of the PeerFlow architecture.

2.2 Authentication

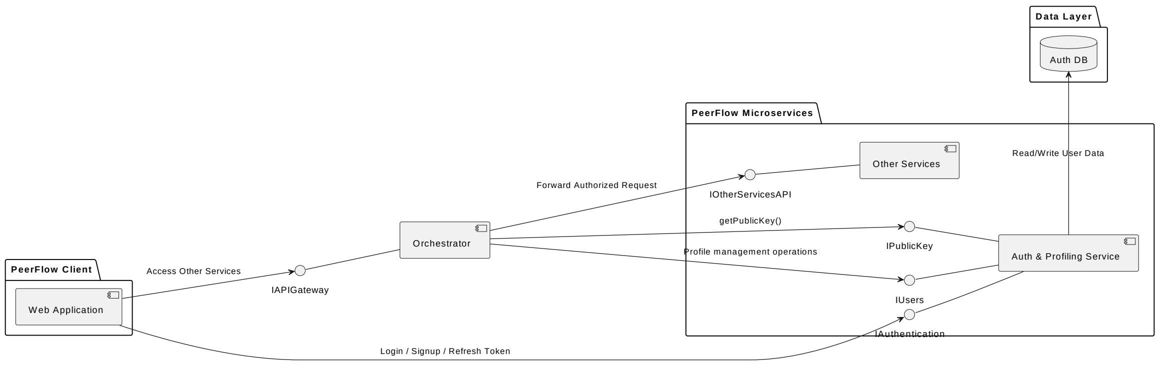

Figure 2.1: C&C view of the authentication process.

2.2.1 Elements

1. **Web Application:** This is the client-side front-end application. It provides the user interface through which users can interact with the PeerFlow system's microservices.

    - **User Login and Signup Form:** It presents a typical login/signup page and sends user credentials to the backend via the Orchestrator. It also handles the storage and refreshing of JWT tokens.

2. **Orchestrator [General Details]:** This service acts as the API Gateway for the PeerFlow system.

3. **Auth & Profiling Service [General Details]:** This microservice is responsible for user authentication and profile management.

    - **IAuthentication interface - Authentication:** Allows the user to sign up, log in, obtain access and refresh JWTs, and refresh the access token.
    - **IPublicKey interface - Public Key Discovery:** Provides an endpoint to retrieve the public key of the Auth & Profiling Service, which can be used to verify its signature on the JWTs. Each JWT is signed with **ES256** asymmetric signature algorithm, which allows other systems to verify the signature without interacting with Auth & Profiling Service.
    - **IUsers interface:** Provides methods to execute CRUD operations on user profiles.

4. **Other Services [General Details]:** This aggregated component represents domain-specific microservices within the PeerFlow system that are not specifically relevant in the view.

2.2.2 Sequence Diagram

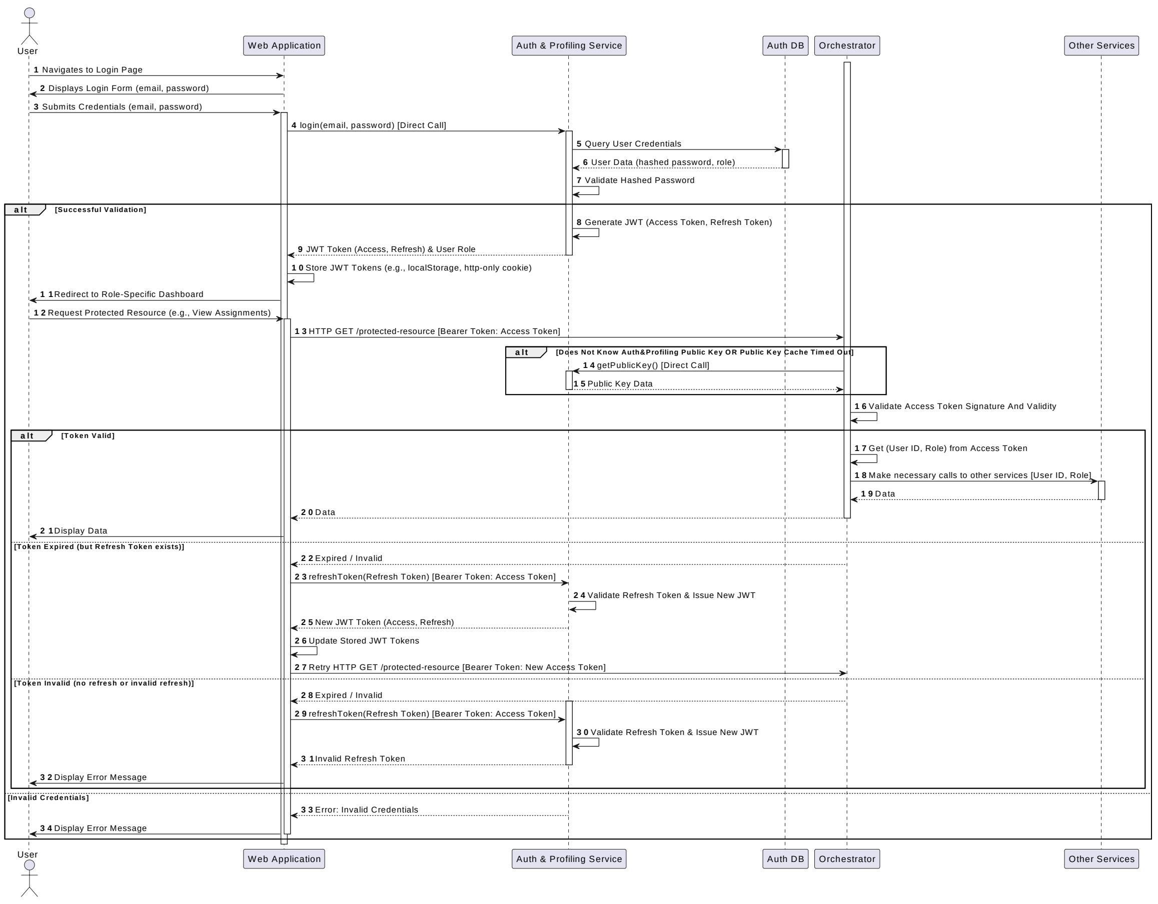
User
Web Application
Auth & Profiling Service
Auth DB
Orchestrator
Other Services
1 Navigates to Login Page
2 Displays Login Form (email, password)
3 Submits Credentials (email, password)
4 login(email, password) [Direct Call]
5 Query User Credentials
6 User Data (hashed password, role)
7 Validate Hashed Password
alt [Successful Validation]
8 Generate JWT (Access Token, Refresh Token)
9 JWT Token (Access, Refresh) & User Role
10 Store JWT Tokens (e.g., localStorage, http-only cookie)
11 Redirect to Role-Specific Dashboard
12 Request Protected Resource (e.g., View Assignments)
13 HTTP GET /protected-resource [Bearer Token: Access Token]
alt [Does Not Know Auth&Profiling Public Key OR Public Key Cache Timed Out]
14 getPublicKey() [Direct Call]
15 Public Key Data
16 Validate Access Token Signature And Validity
alt [Token Valid]
17 Get (User ID, Role) from Access Token
18 Make necessary calls to other services [User ID, Role]
19 Data
20 Data
21 Display Data
[Token Expired (but Refresh Token exists)]
22 Expired / Invalid
23 refreshToken(Refresh Token) [Bearer Token: Access Token]
24 Validate Refresh Token & Issue New JWT
25 New JWT Token (Access, Refresh)
26 Update Stored JWT Tokens
27 Retry HTTP GET /protected-resource [Bearer Token: New Access Token]
[Token Invalid (no refresh or invalid refresh)]
28 Expired / Invalid
29 refreshToken(Refresh Token) [Bearer Token: Access Token]
30 Validate Refresh Token & Issue New JWT
31 Invalid Refresh Token
32 Display Error Message
[Invalid Credentials]
33 Error: Invalid Credentials
34 Display Error Message

Figure 2.2: Sequence diagram of the authentication process.

2.2.3 Rationale

The design choices for the Authentication Flow are justified by the core requirements of security, scalability, and maintainability inherent in a MOOC-like platform, with a specific interaction pattern for authentication:

- **Dedicated Auth & Profiling Service with Clear Interfaces:**
  - **Direct Authentication (Login/Signup/Refresh):** The Web Application directly interacts with the Auth & Profiling Service for user login, signup, and JWT refresh operations. This direct interaction can be optimized for low latency during these critical initial user interactions. `FR-SYS-002` states "The system shall require users to authenticate before accessing role-specific functionalities" and `FR-SYS-003` states "The system shall allow new users (Students or Teachers) to sign up". The Auth & Profiling Service is responsible for handling the user credentials and generating the necessary JWT tokens.
  - **Security:** Centralizing all authentication logic, user management, and JWT token handling within the Auth & Profiling Service ensures that security concerns are isolated and can be managed with specialized expertise and tooling. This aligns with the Security constraint that "The system must ensure the security of user data and

submissions". Explicit interfaces (IAuthentication, IAuthorization, ITokenManagement) clearly define the contract for interactions, enhancing security and reducing potential vulnerabilities.

- **Scalability:** Authentication is a high-traffic operation, especially in MOOCs with a large number of concurrent users. Isolating this functionality allows the Auth & Profiling Service to be independently scaled horizontally (e.g., `QA-PE-4`: Increase in Registered Users specifies maintaining login response times with increased users) without affecting other services.
- **Maintainability:** Changes to authentication mechanisms (e.g., adding multi-factor authentication) or user profile fields (e.g., `QA-MO-01`: Modifying the Student Data Model) are confined to this service, minimizing impact on the rest of the system. The system must support two distinct user roles: Teacher and Student.

- **Orchestrator as an API Gateway Primarily for Authorization Enforcement and Routing to Other Services:**
  - **Unified Entry Point (for non-Auth APIs):** While not handling initial authentication requests directly from WebApp, the Orchestrator still serves as the single entry point for all client requests targeting the "Other Services". This simplifies the Web Application's interaction model for subsequent authorized requests.
  - **Security Enforcement (JWT Verification):** The Orchestrator's primary role in the authentication flow is to verify the JWTs (signed using **ES256** asymmetric signature algorithm) using the public key of the Auth & Profiling Service before forwarding requests to Other Services. This makes the Orchestrator a crucial authorization gate, ensuring that only authenticated and authorized requests reach the downstream services. This means individual microservices (Other Services) do not need to implement their own authorization logic, reducing duplication and potential security vulnerabilities. This also supports the requirement to "enforce access restrictions by granting access only to the system functionalities, data, and user interface views appropriate for the identified user role".

- **Token-Based Authentication (JWT):**
  - **Statelessness:** JWTs are stateless, meaning the server does not need to store session information. This is critical for horizontal scalability, as any instance of a service (both Auth & Profiling Service and Orchestrator for verification) can validate the token without needing to access shared session state. This directly supports the need for horizontal scalability, high availability, and low latency for a large and highly variable number of users.
  - **Decoupling:** JWTs, managed by the Auth & Profiling Service and validated by the Orchestrator, allow for a clear separation of concerns between authentication issuance and authorization enforcement across different components.

- **Dedicated Auth DB:**
  - **Data Isolation:** The Auth DB is exclusively for the Auth & Profiling Service. This aligns with the microservices principle of "data ownership," ensuring that the Auth & Profiling Service has full control over its data schema and evolution.
  - **Performance and Scalability:** This isolation allows the Auth DB to be optimized and scaled independently, which is crucial given the potential for a "gradual but significant increase in the number of active users, profile creation requests, and general platform interactions" and the need to support a "50% increase in concurrent users".

2.3 Assignment Creation

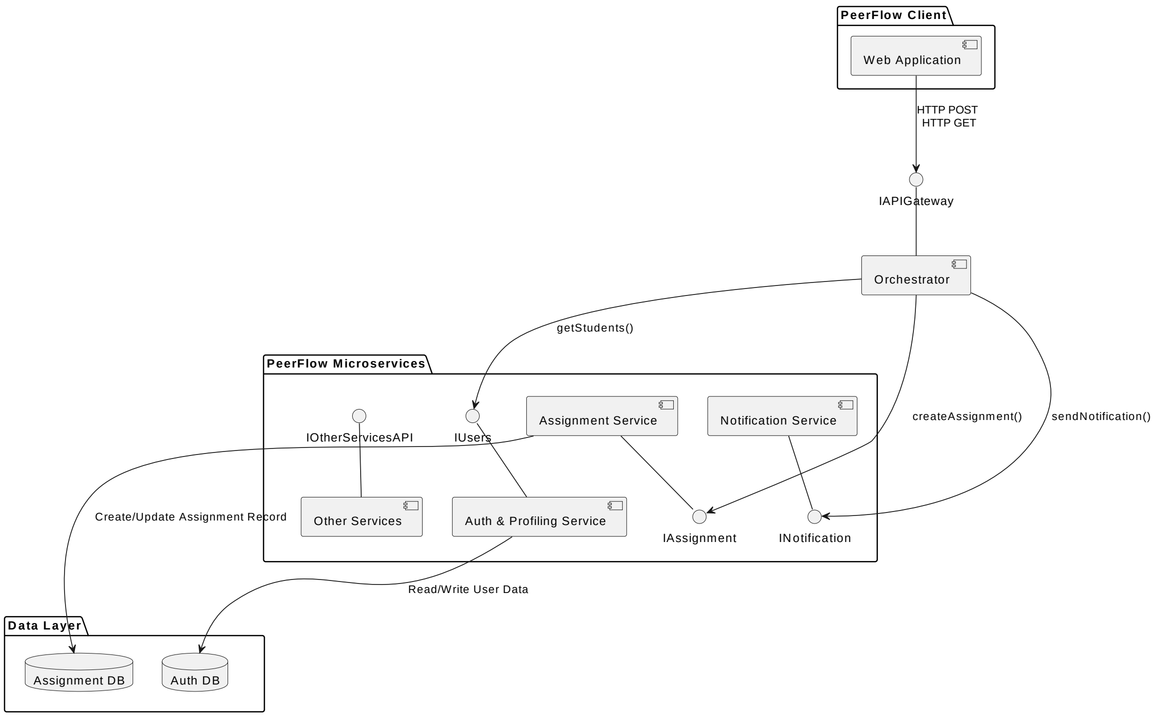

Figure 2.3: C&C view of the assignment creation process.

2.3.1 Elements

1. **Web Application:** This is the client-side front-end application. It provides the user interface through which users can interact with the PeerFlow system's microservices.

    - **Assignment Creation Form:** It provides the user interface through which teachers interact with the PeerFlow system. It presents the teacher with a form or page for creating a new assignment, allowing them to input details like the assignment name, description, deadline, and to select involved students. It initiates the HTTP `GET` request to retrieve the list of available students for selection. It sends the HTTP `POST` request containing all the assignment details, including the selected student IDs, to the Orchestrator.

2. **Orchestrator [General Details]:** This service acts as the API Gateway for the PeerFlow system.

    It is the single entry point for the Web Application to interact with most of the system's microservices.

    - **Student Data Retrieval:** It calls the IUsers interface of the Auth & Profiling Service to fetch the list of registered students, which is then sent back to the Web Application for display in the assignment creation form.
    - **Assignment Creation Orchestration:** It calls the IAssignment interface of the Assignment Service, passing along all the assignment details provided by the teacher, including the selected list of involved student IDs.

- **Notification Trigger:** After the assignment is successfully created, it calls the INotification interface of the Notification Service to send email notifications to the newly involved students about the assignment.

3. **Auth & Profiling Service [General Details]:** This microservice is responsible for user authentication and profile management.

4. **Assignment Service:** This microservice manages the creation, modification, and viewing of assignments.

   - **IAssignment interface - Assignment Creation:** It receives the calls by the Orchestrator containing all the assignment details (name, description, deadline, and the list of involved student IDs). It validates the input (e.g., checking if the deadline is in the future) and generates a new assignment record, associating it with the creating teacher and the specified students in its dedicated data store.
   - **IAssignment interface:** This interface also defines methods for general assignment management, such as viewing or modifying.
   - **Assignment DB:** This is the dedicated database instance for the Assignment Service. It stores all assignment-related data, such as the assignment's name, description, submission deadline, current status, and the list of involved student IDs.

5. **Notification Service [General Details]:** This service exposes a REST API that allows to send email notifications to the users of the system.

6. **Other Services [General Details]:** This aggregated component represents domain-specific microservices within the PeerFlow system that are not specifically relevant in the view.

2.3.2 Sequence Diagram

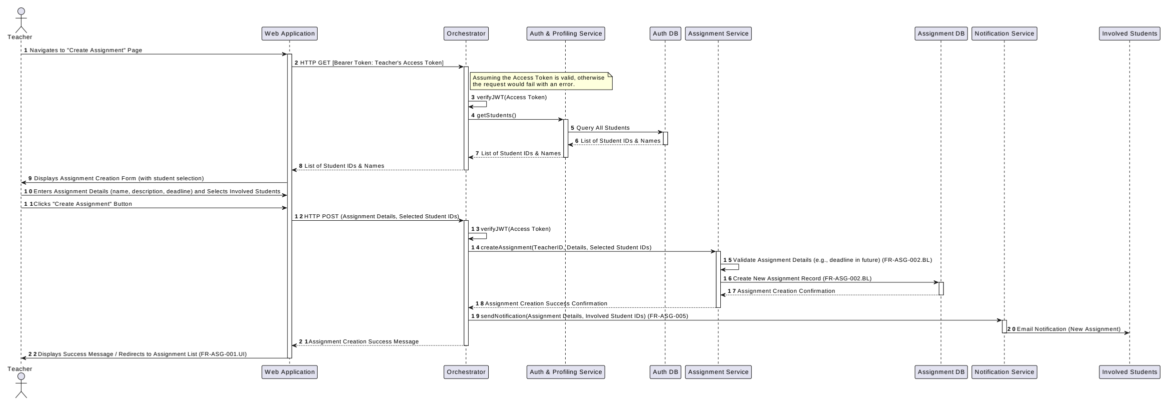
Teacher
Web Application
Orchestrator
Auth & Profiling Service
Auth DB
Assignment Service
Assignment DB
Notification Service
Involved Students
1 Navigates to "Create Assignment" Page
2 HTTP GET [Bearer Token: Teacher's Access Token]
Assuming the Access Token is valid, otherwise the request would fail with an error.
3 verifyJWT(Access Token)
4 getStudents()
5 Query All Students
6 List of Student IDs & Names
7 List of Student IDs & Names
8 List of Student IDs & Names
9 Displays Assignment Creation Form (with student selection)
10 Enters Assignment Details (name, description, deadline) and Selects Involved Students
11 Clicks "Create Assignment" Button
12 HTTP POST (Assignment Details, Selected Student IDs)
13 verifyJWT(Access Token)
14 createAssignment(TeacherID, Details, Selected Student IDs)
15 Validate Assignment Details (e.g., deadline in future) (FR-ASG-002.BL)
16 Create New Assignment Record (FR-ASG-002.BL)
17 Assignment Creation Confirmation
18 Assignment Creation Success Confirmation
19 sendNotification(Assignment Details, Involved Student IDs) (FR-ASG-005)
20 Email Notification (New Assignment)
21 Assignment Creation Success Message
22 Displays Success Message / Redirects to Assignment List (FR-ASG-001.UI)

Figure 2.4: Sequence diagram of the assignment creation process.

2.3.3 Rationale

- **Teacher Role Enforcement and Authorization:**
  - The flow ensures that only authenticated and authorized "Teachers" can create assignments. This is critical for data integrity and system security.
  - The Orchestrator acts as an API Gateway and is responsible for this initial authorization check by locally verifying JWT validity. This centralizes security enforcement, preventing other services from having to re-implement authentication/authorization logic for every request.
  - The Auth & Profiling Service is the single source of truth for user roles and permissions.
- **Efficient Student Involvement Management:**
  - Teachers need to specify "involved students". The flow facilitates this by first allowing the Web Application to fetch a list of available students.
  - By querying the IUsers interface of the Auth & Profiling Service via the Orchestrator, the system ensures that the list of students is always up-to-date and consistent with the central user registry. This avoids data duplication if student data were cached or replicated in other services.
- **Clear Separation of Concerns and Modularity:**
  - The Assignment Service is exclusively responsible for the business logic of creating assignments and managing assignment data (name, description, deadlines, associated students). This adheres to the microservices principle of single responsibility.
  - Validation of assignment details, such as ensuring the "deadline is in the future", is handled directly by the Assignment Service's business logic.
  - Changes to how assignments are created or their data structure can be implemented within the Assignment Service without impacting other parts of the system, supporting the Modifiability non-functional requirement.
- **Decoupled Notification System:**

  - The requirement to ”notify involved students (through email) when a new assignment is created” is handled by the Notification Service.
  - The Orchestrator triggers this notification after the Assignment Service has successfully created the assignment. This loose coupling means the Assignment Service does not need to know the specifics of email sending. It only needs to confirm the assignment's creation.
  - The Notification Service can be independently scaled and can implement robust features like message queues and retry mechanisms to ensure reliable delivery, even if the primary assignment creation process is successful.

- **Scalability and Performance:**
  - The use of dedicated databases (Assignment DB and Auth DB) for each service ensures data autonomy and avoids contention that could arise from a shared database, contributing to better performance and scalability of the overall system.

This flow ensures that assignment creation is a secure, well-orchestrated, and maintainable process within the PeerFlow microservices ecosystem.

2.4 Assignment Submission

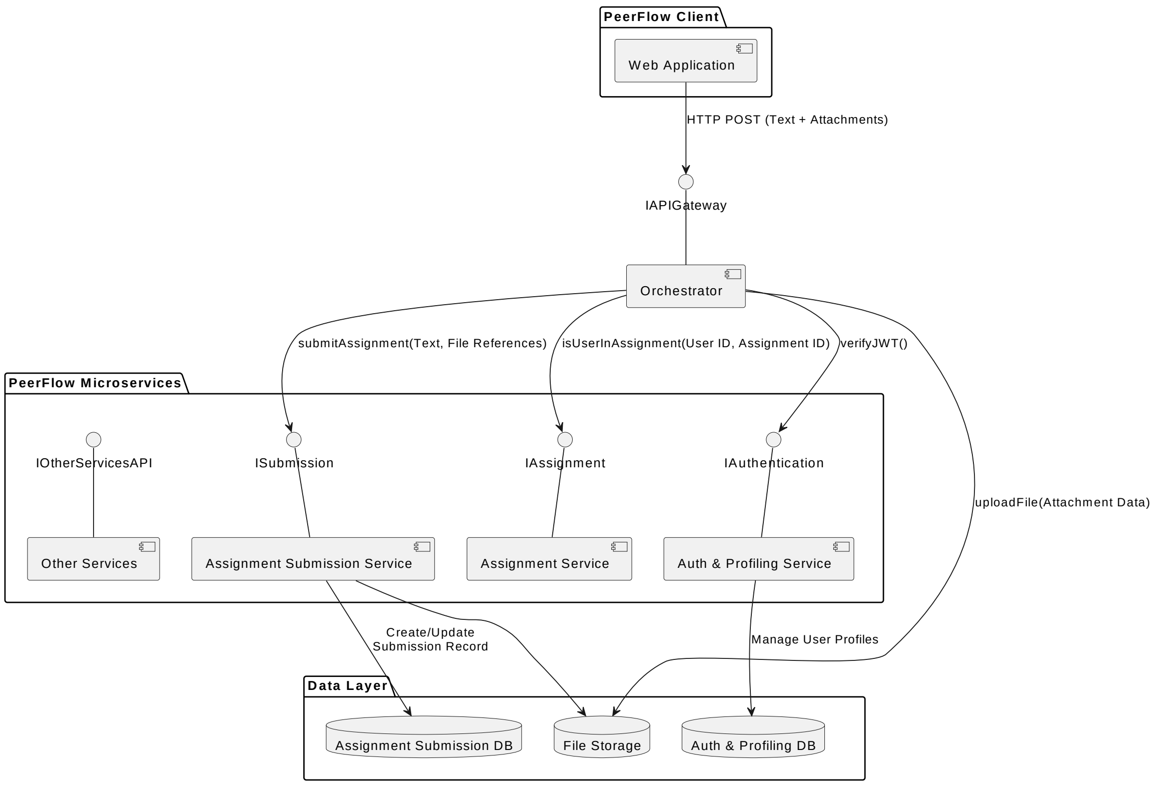

Figure 2.5: C&C view of the assignment submission process.

2.4.1 Elements

1. **Web Application:** This is the client-side front-end application. It provides the user interface through which users can interact with the PeerFlow system's microservices.

   - **Assignment Submission Form:** It shows a submission interface for an assignment, allowing the students to input textual content and (optionally) to upload file attachments (PDF, TXT, JPG). It sends the HTTP `POST` request, containing both the textual input and the file attachment data, to the Orchestrator.

2. **Orchestrator [General Details]:** This service acts as the API Gateway for the PeerFlow system. It is the central coordinator for the submission flow.

   - **File Handling Optimization:** The Orchestrator directly calls the `uploadFile()` operation to upload files on the FileStorageDB (representing the File Storage Service's functionality). This allows to bypass the file transfer through the Assignment Submission Service.
   - **Reference Passing:** After successfully uploading files and obtaining file references (e.g., URLs or unique IDs) from the FileStorageDB, the Orchestrator calls the ISubmission interface of the Assignment Submission Service, finally registering the submission.

3. **Auth & Profiling Service [General Details]:** This microservice is responsible for user authentication and profile management.

4. **Assignment Service:** This microservice manages the creation, modification, and viewing of assignments.

    - **IAssignment interface - Assignment List**: This interface allows the orchestrator to check if the student is assigned and allowed to send a submission.

5. **Assignment Submission Service:** This microservice handles the processing and recording of student submissions.

    - **ISubmission interface:** It receives `submitAssignment()` calls from the Orchestrator. It processes the textual content and the file references received from the Orchestrator. It performs business logic validations (e.g., ensuring submission is before the deadline). It registers the text, file references, and submission timestamp in its dedicated Assignment Submission DB.
    - **Assignment Submission DB:** This is the dedicated database instance for the Assignment Submission Service. It stores submission metadata, including the textual content of the submission, the file references obtained from FileStorageDB, the student's ID, the assignment ID, and the submission timestamp.
    - *Note:* This service is capable of accepting attachments directly, but in this workflow, its ISubmission interface is designed to accept text and file references from the Orchestrator to optimize for higher loads.

6. **Other Services [General Details]:** This aggregated component represents domain-specific microservices within the PeerFlow system that are not specifically relevant in the view.

7. **File Storage DB:** This represents the underlying storage mechanism for file attachments. In this workflow, the Orchestrator directly interacts with it to upload files. It stores the actual file attachments uploaded by students.

2.4.2 Sequence Diagram

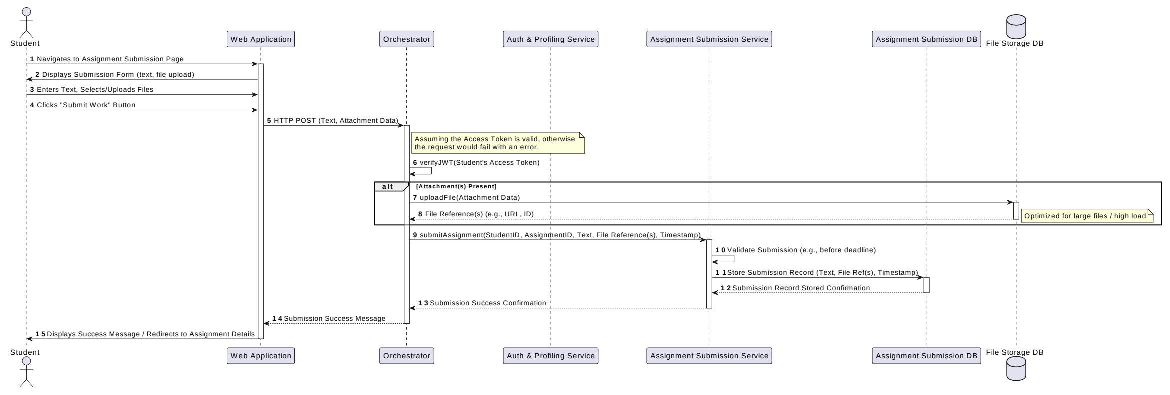
Student
Web Application
Orchestrator
Auth & Profiling Service
Assignment Submission Service
Assignment Submission DB
File Storage DB
1 Navigates to Assignment Submission Page
2 Displays Submission Form (text, file upload)
3 Enters Text, Selects/Uploads Files
4 Clicks "Submit Work" Button
5 HTTP POST (Text, Attachment Data)
Assuming the Access Token is valid, otherwise the request would fail with an error.
6 verifyJWT(Student's Access Token)
alt
[Attachment(s) Present]
7 uploadFile(Attachment Data)
8 File Reference(s) (e.g., URL, ID)
Optimized for large files / high load
9 submitAssignment(StudentID, AssignmentID, Text, File Reference(s), Timestamp)
10 Validate Submission (e.g., before deadline)
11 Store Submission Record (Text, File Ref(s), Timestamp)
12 Submission Record Stored Confirmation
13 Submission Success Confirmation
14 Submission Success Message
15 Displays Success Message / Redirects to Assignment Details

Figure 2.6: Sequence diagram of the assignment submission process.

2.4.3 Rationale

The design of the "Assignment Submission Flow" is optimized to address the critical `QA-PE-3`: Submission Peak requirement, which requires the system to be capable of handling a "large number of students (e.g., 100,000) attempting to submit their assignments concurrently". Therefore, the chosen architecture focuses on high throughput, low error rates, and efficient resource utilization, especially for file attachments.

- **Optimized File Handling for High Loads:**
  - The Orchestrator's direct interaction with the FileStorageDB for attachment uploads is a design choice taken to bypass the Assignment Submission Service from the high load of processing large file streams. This approach ensures that the Assignment Submission Service handles lighter metadata (submission text, file references), allowing it to process a significantly higher volume of submission records efficiently, meeting the "All submission requests are successfully processed within 1 minute of reception" and "The error rate for submissions remains below 0.01%" goals.
  - By assigning file storage to a dedicated and scalable FileStorageDB, resources are utilized by the component best suited for the task. This helps maintain the "average CPU utilization of service instances does not exceed 80% for more than 5 consecutive minutes".
  - The Assignment Submission Service still exposes an interface that accepts files, though it's not used in this scenario. This is compliant with the microservices architecture philosophy.
- **Clear Separation of Concerns (Submission vs. File Storage):**
  - In this case, the Assignment Submission Service focuses solely on managing records of submissions including information like who submitted what, when, and its status, along with references to files. This ensures its business logic remains focused on submission validation and state management.
  - The FileStorageDB is specialized for efficient and scalable storage and retrieval of arbitrary binary data.
- **Authorization and Role Enforcement:**

  - The Orchestrator acts as a security gate, leveraging the IAuthorization interface of the Auth & Profiling Service. This ensures that only authenticated "Students" who are "involved in a specific assignment" can submit their work. This centralized authorization mechanism prevents unauthorized access and maintains data integrity.

- **Transactional Integrity:**
  - The flow implies that a submission is only considered complete once both the file is stored (and a reference obtained) AND the submission record is created in AssignSubmDB. This ensures consistency, even if the actual database operations are distributed.

2.5 Peer Review Start

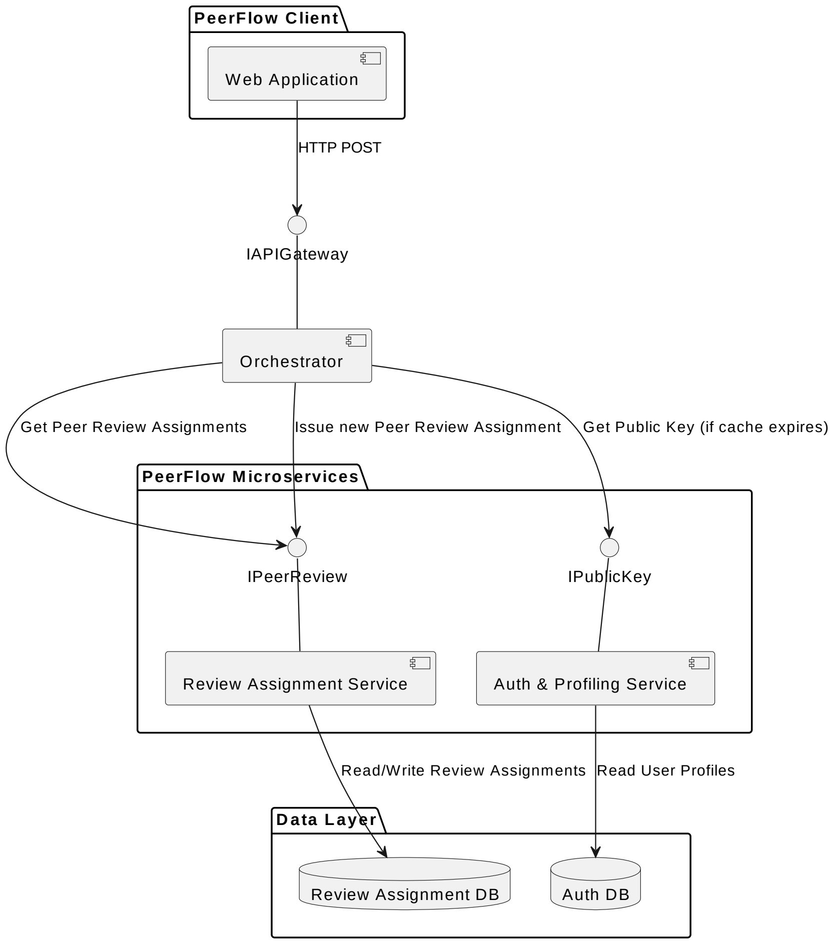

Figure 2.7: C&C view of the peer review start process.

2.5.1 Elements

1. **Web Application:** This is the client-side front-end application. It provides the user interface through which users can interact with the PeerFlow system's microservices.

    - **Peer Review Creation Form:** It shows the user a form which enables input for Rubric creation and Peer Review initialization. It also sends HTTP requests to the Orchestrator to check assignment status, retrieve student lists (if manual pairing), define/edit/delete rubrics, and finally, to start the peer review process.

2. **Orchestrator [General Details]:** This service acts as the API Gateway for the PeerFlow system, coordinating the "Peer Review Start" flow.

    - **Peer Reviews and Rubrics Management:** It forwards the requests related to peer reviews to the IPeerReview interface of the Review Assignment Service. It also checks for correct review pairings.
    - **Notification Trigger:** Upon successful initiation of the peer review process, it triggers notifications via the INotification interface of the Notification Service to inform students that they have been assigned peer reviews.

3. **Auth & Profiling Service [General Details]:** This microservice is responsible for user authentication and profile management.

4. **Review Assignment Service:** This microservice is central to managing the peer review phase.

    - **IPeerReview interface - Peer Review creation:** It receives requests from the Orchestrator to initiate a peer review. This includes setting the number of reviewers, confirming the rubric, specify the pairings.
    - **Review Assignment DB:** This is the dedicated database instance for the Review Assignment Service. It stores defined assessment rubrics (criteria, descriptions, scoring ranges) and the peer review assignment pairings (which student reviews which submission).

5. **Notification Service [General Details]:** This service exposes a REST API that allows to send email notifications to the users of the system.

6. **Other Services [General Details]:** This aggregated component represents domain-specific microservices within the PeerFlow system that are not specifically relevant in the view.

2.5.2 Sequence Diagram

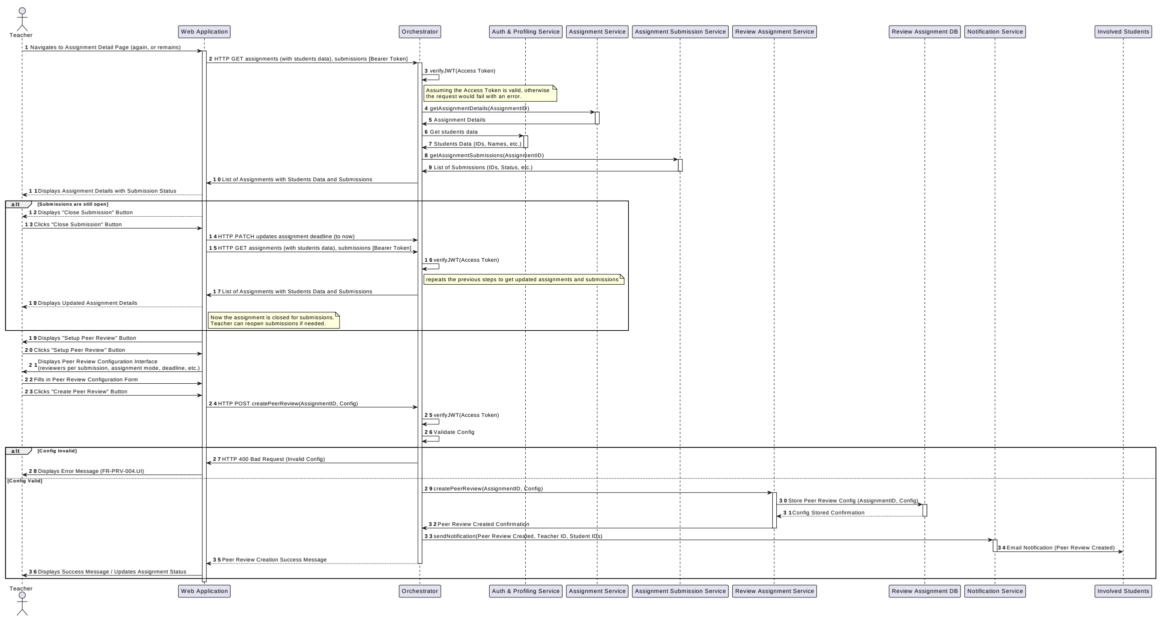
Teacher
Web Application
Orchestrator
Auth & Profiling Service
Assignment Service
Assignment Submission Service
Review Assignment Service
Review Assignment DB
Notification Service
Involved Students
1 Navigates to Assignment Detail Page (again, or remains)
2 HTTP GET assignments (with students data), submissions [Bearer Token]
3 verifyJWT(Access Token)
Assuming the Access Token is valid, otherwise the request would fail with an error.
4 getAssignmentDetails(AssignmentID)
5 Assignment Details
6 Get students data
7 Students Data (IDs, Names, etc.)
8 getAssignmentSubmissions(AssignmentID)
9 List of Submissions (IDs, Status, etc.)
10 List of Assignments with Students Data and Submissions
11 Displays Assignment Details with Submission Status
alt [Submissions are still open]
12 Displays "Close Submission" Button
13 Clicks "Close Submission" Button
14 HTTP PATCH updates assignment deadline (to now)
15 HTTP GET assignments (with students data), submissions [Bearer Token]
16 verifyJWT(Access Token)
repeats the previous steps to get updated assignments and submissions
17 List of Assignments with Students Data and Submissions
18 Displays Updated Assignment Details
Now the assignment is closed for submissions. Teacher can reopen submissions if needed.
19 Displays "Setup Peer Review" Button
20 Clicks "Setup Peer Review" Button
21 Displays Peer Review Configuration Interface (reviewers per submission, assignment mode, deadline, etc.)
22 Fills in Peer Review Configuration Form
23 Clicks "Create Peer Review" Button
24 HTTP POST createPeerReview(AssignmentID, Config)
25 verifyJWT(Access Token)
26 Validate Config
alt [Config Invalid]
27 HTTP 400 Bad Request (Invalid Config)
28 Displays Error Message (FR-PRV-004.UI)
[Config Valid]
29 createPeerReview(AssignmentID, Config)
30 Store Peer Review Config (AssignmentID, Config)
31 Config Stored Confirmation
32 Peer Review Created Confirmation
33 sendNotification(Peer Review Created, Teacher ID, Student IDs)
34 Email Notification (Peer Review Created)
35 Peer Review Creation Success Message
36 Displays Success Message / Updates Assignment Status

Figure 2.8: Sequence diagram of the peer review start process.

2.5.3 Rationale

The ”Peer Review Start” flow is designed to provide teachers with flexible, secure, and reliable means to configure and initiate the peer evaluation process.

- **Teacher Role Enforcement and Authorization:**
  - The Orchestrator serves as a control point, performing JWT verification and role confirmation. This ensures security and prevents unauthorized modifications to assignments or review processes.
  - The Auth & Profiling Service is the single source of truth for user roles and permissions.
- **Conditional Process Flow and Validation:**
  - The system ensures that the preconditions for starting peer review are met.
- **Flexible Peer Review Assignment Mechanisms:**
  - The system supports both ”automatic (random)” and ”manual (defined by the teacher)” peer assignment.
  - The Orchestrator initially fetches data of students, then, when a new Peer Review is being created, it checks for conditions: no students can review their own submission, the number of reviewers per assignment submission must be exactly the one specified by the teacher.
- **Decoupled Notification for Students:**
  - After peer reviews are assigned, students are notified via email.

  - The Orchestrator triggers this notification through the INotification interface of the Notification Service. This separation of concerns ensures that the core peer assignment logic in Review Assignment Service is decoupled from the communication mechanism.

- **Data Integrity and State Management:**
  - The Review Assignment DB is dedicated to storing rubrics and peer review records, ensuring data autonomy and consistency for this critical phase.

2.6 Peer Review Submission

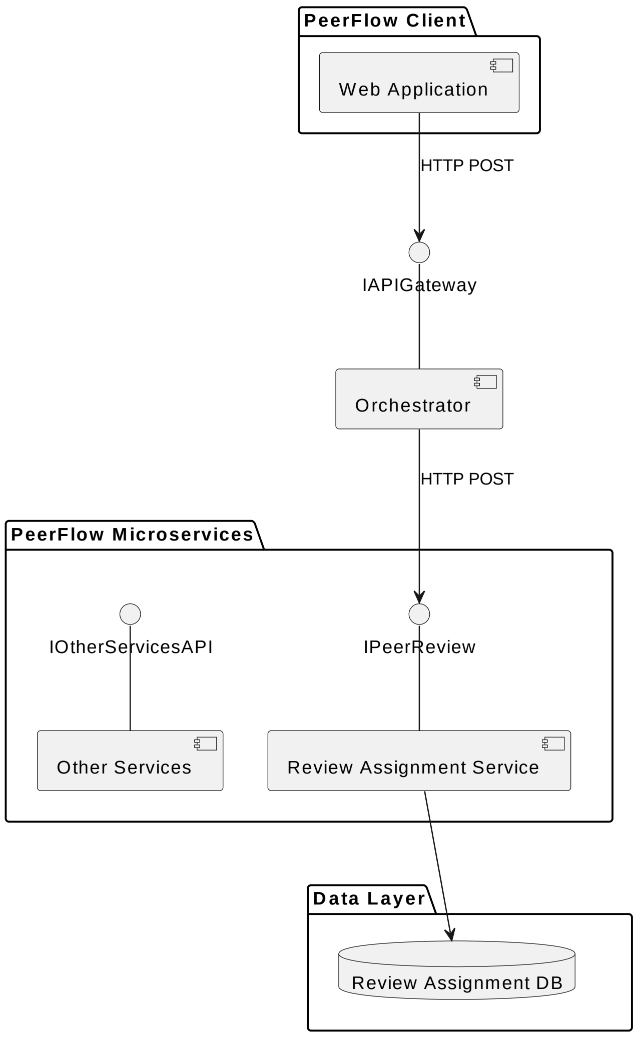

Figure 2.9: C&C view of the peer review submission process.

2.6.1 Elements

1. **Web Application:** This is the client-side front-end application. It provides the user interface through which users can interact with the PeerFlow system's microservices.

   - **Peer Review Submission Form:** It shows a submission interface for a Peer Review, allowing students to view the submission they are reviewing and to input scores and feedback for each criterion. Also, displays the data of the submission under review.

2. **Orchestrator [General Details]:** This service acts as the API Gateway for the PeerFlow system, coordinating the "Peer Review Start" flow.

3. **Auth & Profiling Service [General Details]:** This microservice is responsible for user authentication and profile management.

4. **Review Assignment Service:** This microservice is central to managing the peer review phase, including the assignment pairings, rubrics, and now, the processing and storage of completed reviews.

   - **IPeerReview interface - Peer Review Submission:** It receives data from the Orchestrator and registers the Peer Review in DB.
   - **Review Assignment DB:** This is the dedicated database instance for the Review Assignment Service. It stores peer review assignment pairings (which student reviews which submission), the assessment rubrics, and now also the completed peer review records (scores, textual justifications, reviewer identity, original author, timestamp).

5. **Other Services [General Details]:** This aggregated component represents domain-specific microservices within the PeerFlow system that are not specifically relevant in the view.

6. **File Storage DB:** This represents the underlying storage mechanism for file attachments. In this workflow, the Orchestrator directly interacts with it to download files.

2.6.2 Sequence Diagram

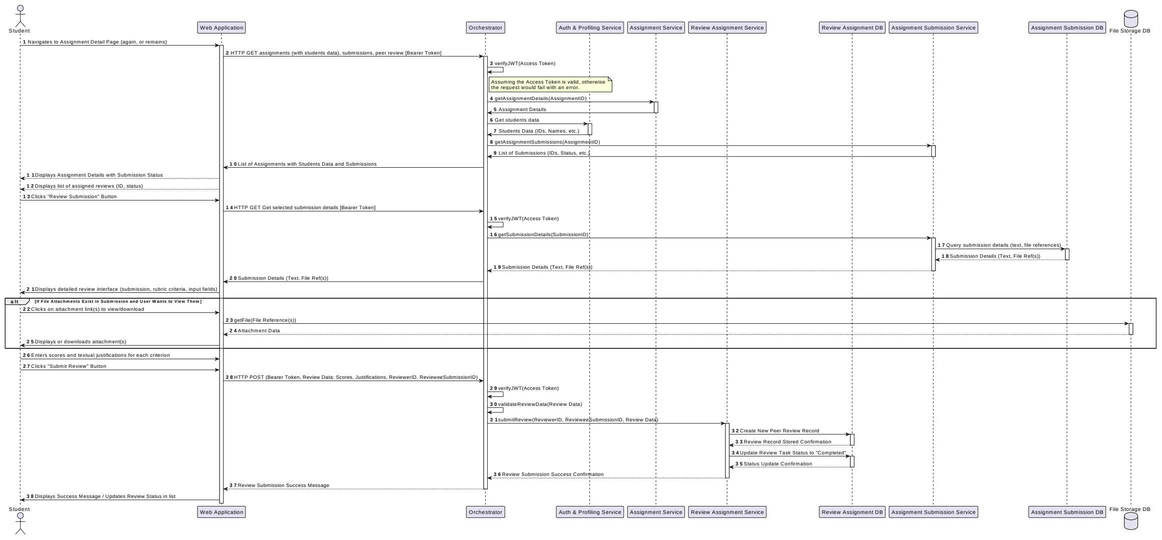
Student
Web Application
Orchestrator
Auth & Profiling Service
Assignment Service
Review Assignment Service
Review Assignment DB
Assignment Submission Service
Assignment Submission DB
File Storage DB
1 Navigates to Assignment Detail Page (again, or remains)
2 HTTP GET assignments (with students data), submissions, peer review [Bearer Token]
3 verifyJWT(Access Token)
Assuming the Access Token is valid, otherwise the request would fail with an error.
4 getAssignmentDetails(AssignmentID)
5 Assignment Details
6 Get students data
7 Students Data (IDs, Names, etc.)
8 getAssignmentSubmissions(AssignmentID)
9 List of Submissions (IDs, Status, etc.)
10 List of Assignments with Students Data and Submissions
11 Displays Assignment Details with Submission Status
12 Displays list of assigned reviews (ID, status)
13 Clicks "Review Submission" Button
14 HTTP GET Get selected submission details [Bearer Token]
15 verifyJWT(Access Token)
16 getSubmissionDetails(SubmissionID)
17 Query submission details (text, file references)
18 Submission Details (Text, File Ref(s))
19 Submission Details (Text, File Ref(s))
20 Submission Details (Text, File Ref(s))
21 Displays detailed review interface (submission, rubric criteria, input fields)
alt
[If File Attachments Exist in Submission and User Wants to View Them]
22 Clicks on attachment link(s) to view/download
23 getFile(File Reference(s))
24 Attachment Data
25 Displays or downloads attachment(s)
26 Enters scores and textual justifications for each criterion
27 Clicks "Submit Review" Button
28 HTTP POST (Bearer Token, Review Data: Scores, Justifications, ReviewerID, RevieweeSubmissionID)
29 verifyJWT(Access Token)
30 validateReviewData(Review Data)
31 submitReview(ReviewerID, RevieweeSubmissionID, Review Data)
32 Create New Peer Review Record
33 Review Record Stored Confirmation
34 Update Review Task Status to "Completed"
35 Status Update Confirmation
36 Review Submission Success Confirmation
37 Review Submission Success Message
38 Displays Success Message / Updates Review Status in list

Figure 2.10: Sequence diagram of the peer review submission process.

2.6.3 Rationale

- **Student Interface:**
  - The flow begins with the Web Application displaying a list of assigned reviews to the student. When a student selects a submission, the Web Application presents a detailed review interface including the peer's submitted text and attachments. This ensures students have all the necessary information to conduct a good review.

- **Orchestrator as Intelligent Coordinator:**
  - The Orchestrator acts as the coordinator for all interactions between the Web Application and the other services involved. This simplifies the client-side logic, as the Web Application only communicates with one API Gateway (IAPIGateway).
  - **Authorization Enforcement:** At every step, the Orchestrator performs authorization checks. This ensures that only authenticated students can access their assigned reviews and that they are reviewers for that specific submission.
  - **Data Aggregation for UI:** The Orchestrator is responsible for aggregating data from multiple microservices before presenting it to the Web Application. This offloads complexity from the frontend and ensures a coherent data model for display.
  - **File Retrieval:** The Web Application retrieves file attachments from the File Storage DB using references obtained from the Assignment Submission Service. This pattern is efficient for handling large files, ensuring that the Assignment Submission Service only handles metadata and doesn't become a bottleneck for data transfer.

- **Dedicated Services for Core Responsibilities:**
  - **Review Assignment Service for Review Data Management:** This service manages review assignments, rubrics, and also the processing and storage of completed peer review records. It provides interfaces that encapsulate all review-specific business logic. Validations like checking scores against rubric ranges and requiring justifications are performed here.

  - **Assignment Submission Service for Submission Content:** This service remains the authoritative source for the content of student submissions (text and file references). This separation maintains a clear boundary between the "submission" and "review" domains, addressing the separation of concerns principle.
  - **File Storage DB for Efficient File Access:** Its direct interaction with the Orchestrator for `getFile()` calls ensures fast and scalable retrieval of attachments.

- **Data Integrity and State Management:**
  - The Review Assignment DB is the single source for peer review assignments, rubrics, and the completed review records. This ensures consistency across the system regarding the status and content of reviews.

2.7 Peer Review Results Aggregation

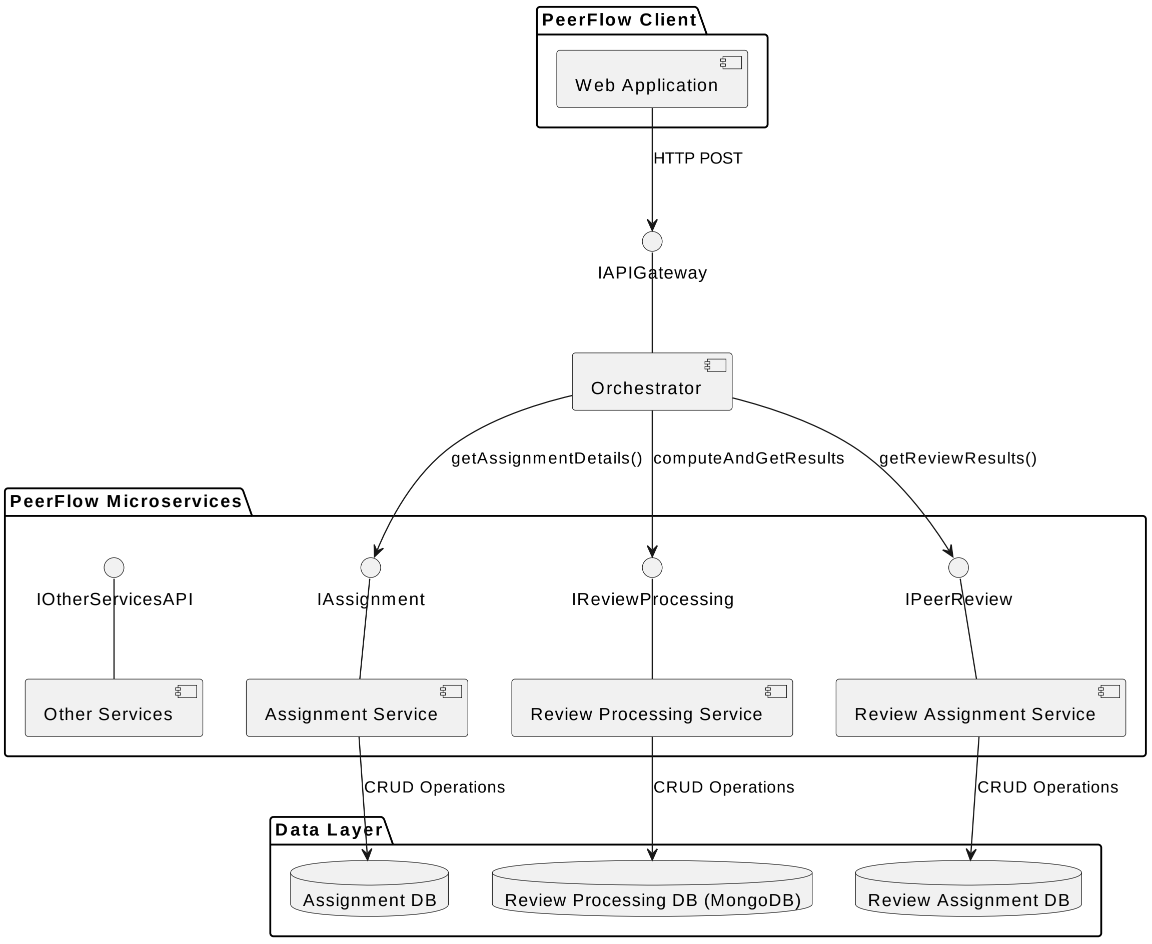

Figure 2.11: C&C view of the peer review results aggregation process.

2.7.1 Elements

1. **Web Application:** This is the client-side front-end application. It provides the user interface through which users can interact with the PeerFlow system's microservices.

   - **Start Aggregation Button:** The Web Application presents the teacher with a button to manually initiate the review aggregation process for a specific assignment. It sends the HTTP `POST` request to the Orchestrator when the teacher triggers the aggregation.

2. **Orchestrator [General Details]:** This service acts as the API Gateway for the PeerFlow system, coordinating the "Peer Review Start" flow.

   - **Data Extraction Orchestration:** Instead of the Review Processing Service extracting data directly, the Orchestrator now orchestrates the extraction. It calls the IAssignment interface of the Assignment Service and the IPeerReview interface of the Review Assignment Service to check that the requesting teacher is the creator of the assignment and to gather all the necessary raw data (completed reviews, rubrics, and pairings) for the specific assignment.
   - **Aggregation Trigger:** After extracting the raw data, the Orchestrator calls the IReviewProcessing interface of the Review Processing Service, passing this extracted raw review data to it to initiate the aggregation calculation.

3. **Review Assignment Service:** This microservice is responsible for managing peer review assignments, rubrics, and storing the raw, individual completed review records submitted by students.

   - **IPeerReview interface - Raw Data Provider:** It provides the essential interfaces that the Orchestrator (now acting as the extractor) uses to retrieve the necessary raw data for aggregation.
   - **Review Assignment DB:** This is the dedicated database instance for the Review Assignment Service. It serves as the source of raw data for the aggregation process, storing individual completed peer review records, the assessment rubrics, and the peer review assignment pairings.

4. **Review Processing Service:** This is the dedicated microservice performing the ETL (Extract, Transform, Load) process for peer review results. Its core function is to compute and store aggregated statistics from the raw review data.

   - **IPeerReview interface - Aggregation Initiation:** It receives the raw review data (extracted by the Orchestrator) and executes the calculation.
   - **Transformation:** It performs the analytics and calculations to derive the aggregated results (e.g., overall average scores, per-criterion averages, score distributions for assignments, and aggregated scores/counts per submission and per reviewer) as defined by your AggregatedByAssignmentResult, AggregatedBySubmissionResult, and AggregatedByReviewResult data views.
   - **Loading:** Once computed, it stores these aggregated results into its dedicated Review Processing DB (MongoDB).
   - **Review Processing DB:** This is the dedicated database instance for the Review Processing Service, explicitly identified as MongoDB to suit the flexible and analytical nature of aggregated results. It stores the pre-computed, aggregated results:

AggregatedByAssignmentResult, AggregatedBySubmissionResult, and AggregatedByReviewResult objects. These are optimized for fast querying and reporting.

5. **Assignment Service:** This microservice manages assignment details, including their status.
    - **IAssignment interface - Status Update:** The Orchestrator uses this interface to retrieve all necessary assignment data.
    - **Assignment DB:** This is the dedicated database instance for the Assignment Service. It stores assignment details.
6. **Other Services [General Details]:** This aggregated component represents domain-specific microservices within the PeerFlow system that are not specifically relevant in the view.

2.7.2 Sequence Diagram

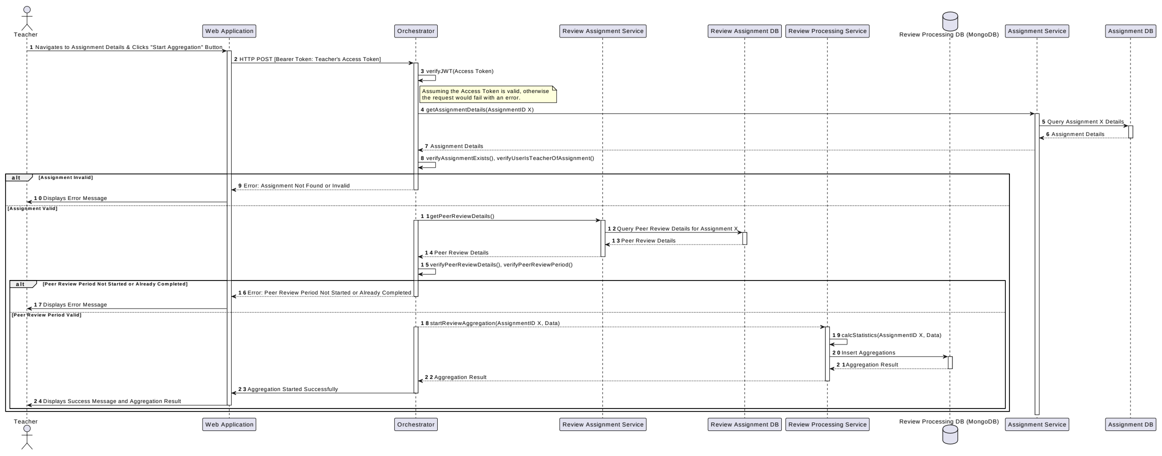
Teacher
Web Application
Orchestrator
Review Assignment Service
Review Assignment DB
Review Processing Service
Review Processing DB (MongoDB)
Assignment Service
Assignment DB
1 Navigates to Assignment Details & Clicks "Start Aggregation" Button
2 HTTP POST [Bearer Token: Teacher's Access Token]
3 verifyJWT(Access Token)
Assuming the Access Token is valid, otherwise the request would fail with an error.
4 getAssignmentDetails(AssignmentID X)
5 Query Assignment X Details
6 Assignment Details
7 Assignment Details
8 verifyAssignmentExists(), verifyUserIsTeacherOfAssignment()
alt [Assignment Invalid]
9 Error: Assignment Not Found or Invalid
10 Displays Error Message
[Assignment Valid]
11 getPeerReviewDetails()
12 Query Peer Review Details for Assignment X
13 Peer Review Details
14 Peer Review Details
15 verifyPeerReviewDetails(), verifyPeerReviewPeriod()
alt [Peer Review Period Not Started or Already Completed]
16 Error: Peer Review Period Not Started or Already Completed
17 Displays Error Message
[Peer Review Period Valid]
18 startReviewAggregation(AssignmentID X, Data)
19 calcStatistics(AssignmentID X, Data)
20 Insert Aggregations
21 Aggregation Result
22 Aggregation Result
23 Aggregation Started Successfully
24 Displays Success Message and Aggregation Result

Figure 2.12: Sequence Diagram of the peer review results aggregation process.

2.7.3 Rationale

The "Peer Review Results Aggregation Flow" is designed as a distinct ETL (Extract, Transform, Load) process, addressing the requirements for "Aggregation and visualization of evaluation results for students and teachers" and "Accessing detailed evaluation reports for teachers".

- **Teacher-Initiated Processing:**
    - Teachers are in direct control over when aggregation occurs, allowing them to finalize reviews and verify conditions before results are processed and released.
    - The Web Application exposes a "Start Aggregation" button, which sends a request to the Orchestrator.
- **Orchestrator as Coordinator:**
    - The Orchestrator acts as the central coordinator, bridging the Web Application with the backend services for aggregation.

  - **Authorization:** It first authorizes the teacher's request, verifying their JWT token and role to ensure only authorized teachers can initiate this process for their assignments.
  - **Data Extraction:** Crucially, the Orchestrator now takes direct responsibility for extracting all necessary raw data from the Review Assignment Service. It calls IPeerReview to retrieve completed reviews, rubric details, and peer review pairings. This consolidates data retrieval logic at the API Gateway level.
  - **Aggregation Trigger:** After successfully extracting the raw data, the Orchestrator then explicitly triggers the Review Processing Service (IReviewProcessing), passing the extracted raw data to it. This design ensures that the Review Processing Service receives precisely the data it needs to perform its calculations.

- **Review Assignment Service as Authoritative Raw Data Source:**
  - Review Assignment Service remains the single source for all raw review-related data: individual completed reviews, assessment rubrics, and the peer review assignment pairings.
  - The Orchestrator explicitly reads this data from Review Assignment Service's interfaces, ensuring consistency and preventing redundant data storage elsewhere. This aligns with the microservices principle of data ownership.

- **Review Processing Service for Dedicated Analytical Processing:**
  - The role of the Review Processing Service is highly specialized: it performs the calculation of statistics ("Transform" phase) and stores the results in its database ("Load" phase).
  - By receiving pre-extracted data from the Orchestrator, the Review Processing Service can focus on its analytical task, which is optimized for fast read access for reporting.
  - The use of Review Processing DB (MongoDB) is ideal for storing flexible, aggregated analytical results, enabling fast querying for visualization.

3 Allocation Views

3.1 Deployment Diagram

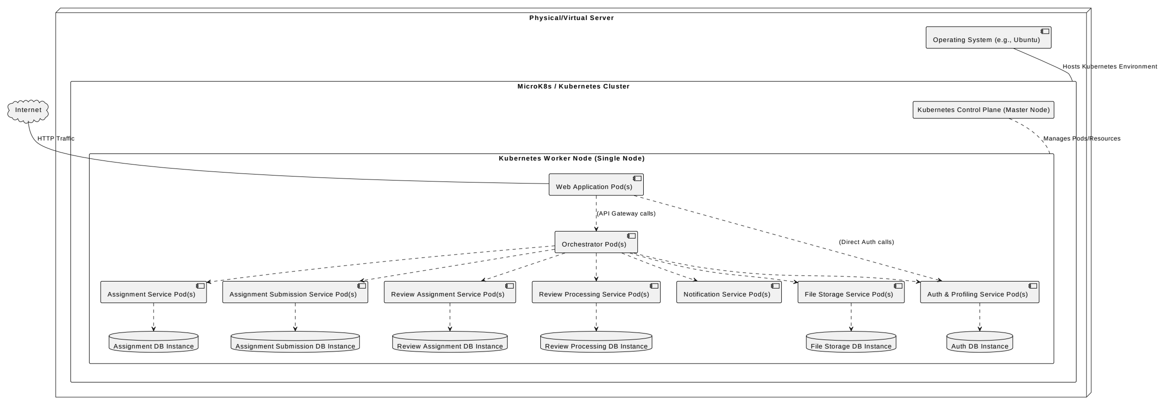
Physical/Virtual Server
Operating System (e.g., Ubuntu)
Hosts Kubernetes Environment
MicroK8s / Kubernetes Cluster
Kubernetes Control Plane (Master Node)
Manages Pods/Resources
Internet
HTTP Traffic
Kubernetes Worker Node (Single Node)
Web Application Pod(s)
(API Gateway calls)
(Direct Auth calls)
Orchestrator Pod(s)
Assignment Service Pod(s)
Assignment Submission Service Pod(s)
Review Assignment Service Pod(s)
Review Processing Service Pod(s)
Notification Service Pod(s)
File Storage Service Pod(s)
Auth & Profiling Service Pod(s)
Assignment DB Instance
Assignment Submission DB Instance
Review Assignment DB Instance
Review Processing DB Instance
File Storage DB Instance
Auth DB Instance

Figure 3.1: Deployment diagram of the system.

The system's deployment architecture is centered around a single Ubuntu Virtual Machine (VM). This VM serves as the host for a single-node Kubernetes cluster, powered by MicroK8s. Within this Kubernetes environment, the entire system is deployed, encompassing the Web Application, an Orchestrator, all defined microservices, and their respective database instances.

For enhanced scalability and availability, each microservice is deployed as a series of Kubernetes Pods, with more than one replica for each service. This allows the system to handle increased loads and ensures continuous operation even if one instance fails.

To prevent data inconsistency issues, the database instances themselves are not replicated. Instead, each database instance is designed to manage its own scalability, leveraging technologies like MongoDB, which inherently supports horizontal scalability and fault tolerance.

The Kubernetes cluster also includes a Master Node, responsible for managing the deployment and lifecycle of the Pods within the cluster.

3.1.1 Rationale

The decision to deploy PeerFlow on a single-node Kubernetes cluster utilizing MicroK8s on an Ubuntu Virtual Machine was primarily driven by the immediate availability of the resources. This lightweight setup provides a practical environment for demonstrating the system's microservices architecture and core functionalities. However, it is crucial to acknowledge that for a system designed to support a large and highly variable number of users, typical of MOOC environments like Coursera or Udacity, this single-node configuration represents a significant constraint. Ideally, a high-performance, multi-node Kubernetes cluster would be essential. Such a cluster would offer the necessary computational and storage resources, along with enhanced fault tolerance and horizontal scaling capabilities, to effectively manage the demands of potentially millions of concurrent users and a vast amount of data, ensuring optimal performance and availability under peak loads.

4 Selected Technologies

This chapter details the key technologies chosen for this project, explaining the rationale behind each decision. The main driver of our decisions is a commitment to performance, scalability, availability, fast development, and long-term maintainability. The following sections will elaborate on each chosen technology, outlining its role within the architecture and the specific advantages it brings to the overall solution.

4.1 Backend Framework: FastAPI

FastAPI is a modern, high-performance web framework for building APIs with Python based on standard Python type hints. In PeerFlow, FastAPI is used to develop the REST APIs for each of the backend microservices.

4.1.1 Key Advantages

- **High Performance:** FastAPI is built on top of Starlette, a lightweight ASGI framework for building async web services, and Pydantic, a fast and extensible data validation library, making it one of the fastest Python frameworks available, on par with NodeJS and Go. This is crucial for handling a large and variable number of users in a MOOC environment. FastAPI is also based on the OpenAPI standard for API creation, focused on understandability from both human and machine without requiring direct access to the source code or additional documentation.
- **Fast to Code:** It is designed to be easy to use and learn, with great editor support and autocompletion, significantly increasing development speed. Features like type hints reduce human-induced errors, reducing debuggin time. This supports the project's need for rapid development.
- **Automatic Data Validation and Serialization:** Leveraging Pydantic, FastAPI provides automatic request data validation, serialization, and deserialization, ensuring data integrity.
- **Automatic API Documentation:** It automatically generates interactive API documentation starting from OpenAPI definitions, using Swagger UI for interactive exploration of the interfaces, and ReDoc for a more detailed and structured presentation, which is invaluable for a microservices architecture where multiple services interact.

FastAPI was chosen due to its excellent performance characteristics, which are essential for a system designed for MOOC environments with potentially high user loads. Its support for rapid development and automatic data validation aligns with the project's need to efficiently build and maintain multiple robust microservices. The auto-generated documentation also aids in managing the complexity of inter-service communication and improving the understandability of the system.

4.2 Frontend Framework: Vue.js

Vue.js is a progressive JavaScript framework for building user interfaces. It is built on top of HTML, CSS and JavaScript and provides a declarative, component-based programming model to develop UIs of any complexity. In the PeerFlow project, Vue.js is utilized to create the client-side Web Application, providing an interface for both Students and Teachers to interact with the system's functionalities.

4.2.1 Key Advantages

- **Approachable and Easy to Integrate:** Being based on HTML, CSS and JavaScript, Vue.js is easy to learn for developers familiar with classic frontend development. It also provides and extensive documentation.
- **Component-Based Architecture:** Its component-based programming model, leads to a frontend code that is more modular, maintainable, and scalable. This aligns with the microservices architecture of the backend.
- **Performance:** Vue.js is known for its good performance, due to its reactive and compiler-optimized rendering system.
- **Rich Ecosystem:** It has a comprehensive ecosystem of libraries and tools, facilitating the development of web applications of various form and scale.

Vue.js was selected to develop the User Interface (UI) for PeerFlow mainly because of its component-based architecture, whose modularity is a good fit for interacting with a microservices backend, and for its performance and ease of use. These advantages contribute to the overall goal of creating an efficient and user-friendly platform.

4.3 DBMS: MongoDB

MongoDB is document-based DBMS, classified as a NoSQL database. In PeerFlow, MongoDB is specifically used as the dedicated database for the Review Processing Service to store aggregated review results.

4.3.1 Key Advantages

- **Flexible Schema:** Document-based storage allows for flexible and evolving data structures, which is ideal for storing data associated to continously evolving microservices.
- **Scalability:** MongoDB is designed for horizontal scalability, allowing it to handle large volumes of data and high throughput, which is beneficial for MOOC-scale applications.
- **Rich Query Language:** It provides a powerful query language for accessing and analyzing data stored in documents. This is beneficial for generating reports and visualizations.

- **Performance for Specific Use Cases:** For read-heavy analytical workloads on denormalized data, MongoDB can offer high performance.

MongoDB was chosen for the DB of the microservices mainly due to its high horizontal scalability, necessary to handle the high number of requests in a MOOC environment, and its suitability for storing and querying for aggregated results, such as overall assignment statistics and detailed reports. Its schema flexibility is also useful to seamlessly integrate new functionalities for both students and teachers.

4.4 File storage system: SeaweedFS

SeaweedFS is an open-source distributed object store and file system designed for efficiently handling a large number of files of any dimension. In PeerFlow, SeaweedFS acts as the File Storage Service, managing all file attachments submitted by students for their assignments.

4.4.1 Key Advantages

- **Efficient Handling of Many Files:** Optimized for storing and serving billions of files quickly.
- **Scalability and High Availability:** It can be easily scaled out to accommodate growing storage needs and provide high availability.
- **Lower Overhead:** Compared to traditional distributed file systems, it often has lower per-file overhead, making it cost-effective for large quantities of files.
- **REST API:** Provides a simple HTTP REST API for file operations, making it easy to integrate with other defined microservices like the Orchestrator or Assignment Submission Service.

SeaweedFS was selected to manage student file submissions in a scalable and efficient manner. This is crucial for handling potential submission peaks `QA-PE-3` and offloading the Assignment Submission Service by allowing the Orchestrator to interact directly with the file storage for uploads. This design contributes to the overall performance and robustness of the submission process.

4.5 Virtualization: Docker and Kubernetes

Docker is a containerization platform used for developing, shipping, and running applications by packaging them into units called containers. Kubernetes is an open-source container orchestration tool designed to automate the deployment, scaling, and management of these containerized applications. In PeerFlow, Docker is employed to containerize each microservice and their respective databases (DBs), ensuring consistent environments across development, testing, and production stages. Kubernetes is then utilized to orchestrate these containerized components within a cluster, managing their deployment, scaling to meet user demand, and ensuring high availability. Each microservice is allocated to a Kubernetes pod, which in turn manages the replicas of its microservice.

4.5.1 Key Advantages

- **Consistent Environments (Docker):** Containers encapsulate an application and its dependencies, ensuring that it runs identically and reliably regardless of the underlying infrastructure.
- **Isolation (Docker):** Docker ensures that microservices run in isolated environments. This isolation reduces inter-service conflicts, and improves modifiability by allowing changes within one service with minimal impact on others (supporting scenarios like `QA-MO-01` and `QA-MO-2`). This also aligns with the microservices principle of independent deployment.
- **Horizontal Scaling and High Availability (Kubernetes):** Kubernetes can automatically scale service instances horizontally based on demand, critical for supporting a large and variable number of users typical of MOOC environments. It also provides a self-healing capability by comparing the desided state (e.g. the number of replicas, specified by the developer) to the current state and acting upon detected differences (as per `QA-AV-1`). Moreover Kubernetes can scale service instances based on resource consumption like the Assignment Service during submission peaks (`QA-PE-3`), and balance the load between replicas.
- **Automated Deployment and Management (Kubernetes):** Kubernetes automates complex tasks such as application rollouts and rollbacks, which is essential for maintaining system stability and supports deployability requirements like deploying bug fixes with zero downtime (`QA-DE-1`) or rolling back malfunctioning services (`QA-DE-2`). It also provides service discovery, for instance, by leveraging health check endpoints for automated monitoring within the cluster (`QA-IN-2`).

Docker and Kubernetes are pratically necessary technologies to satisfy the system's requirements of high availability and scalability, dictated by the necessities of a MOOC environment, which includes traffic spikes and rapid user growth. Docker and Kubernetes directly support several key non-functional requirements:

- Modifiability, by ensuring services are isolated and can be updated independently.
- Deployability, by facilitating automated CI/CD pipelines and enabling scenarios such as deploying a bug fix to a specific service (`QA-DE-1`) and rolling back a malfunctioning service (`QA-DE-2`) efficiently.
- Availability, by managing service instance failures and ensuring continuous operation (`QA-AV-1`).

The entire PeerFlow system, including its web application, orchestrator, microservices, and their databases, is designed to be deployed within this Kubernetes cluster environment.

4.6 Testing: Pytest, Postman and Newman

Pytest is a Python testing framework that allows writing simple, scalable test cases. Postman is a collaboration platform for API development, often used for designing, building, testing, and documenting APIs. Newman is a command-line Collection Runner for Postman, enabling the automation of API tests. These tools are used in PeerFlow to implement automated testing.

4.6.1 Key Advantages

- **Comprehensive Testing (Pytest):** Enables robust unit and integration testing for Python-based backend services.
- **API Design and Manual Testing (Postman):** Provides an intuitive GUI for designing, debugging, and manually testing REST APIs.
- **Automated API Testing (Newman):** Allows Postman collections to be run from the command line, facilitating integration into CI/CD pipelines for automated API end-to-end or integration testing.
- **Improved Software Quality:** Collectively, these tools help ensure the reliability and correctness of individual services and their interactions within the microservices architecture.

Pytest, Postman, and Newman were chosen to design, implement and automate tests. Pytest supports backend unit tests for the Python-based microservices. Postman is used to define integration and load tests of entire system. Automating these tests via Newman in a CI/CD pipeline ensures continuous quality assurance.

List of Figures

List of Tables