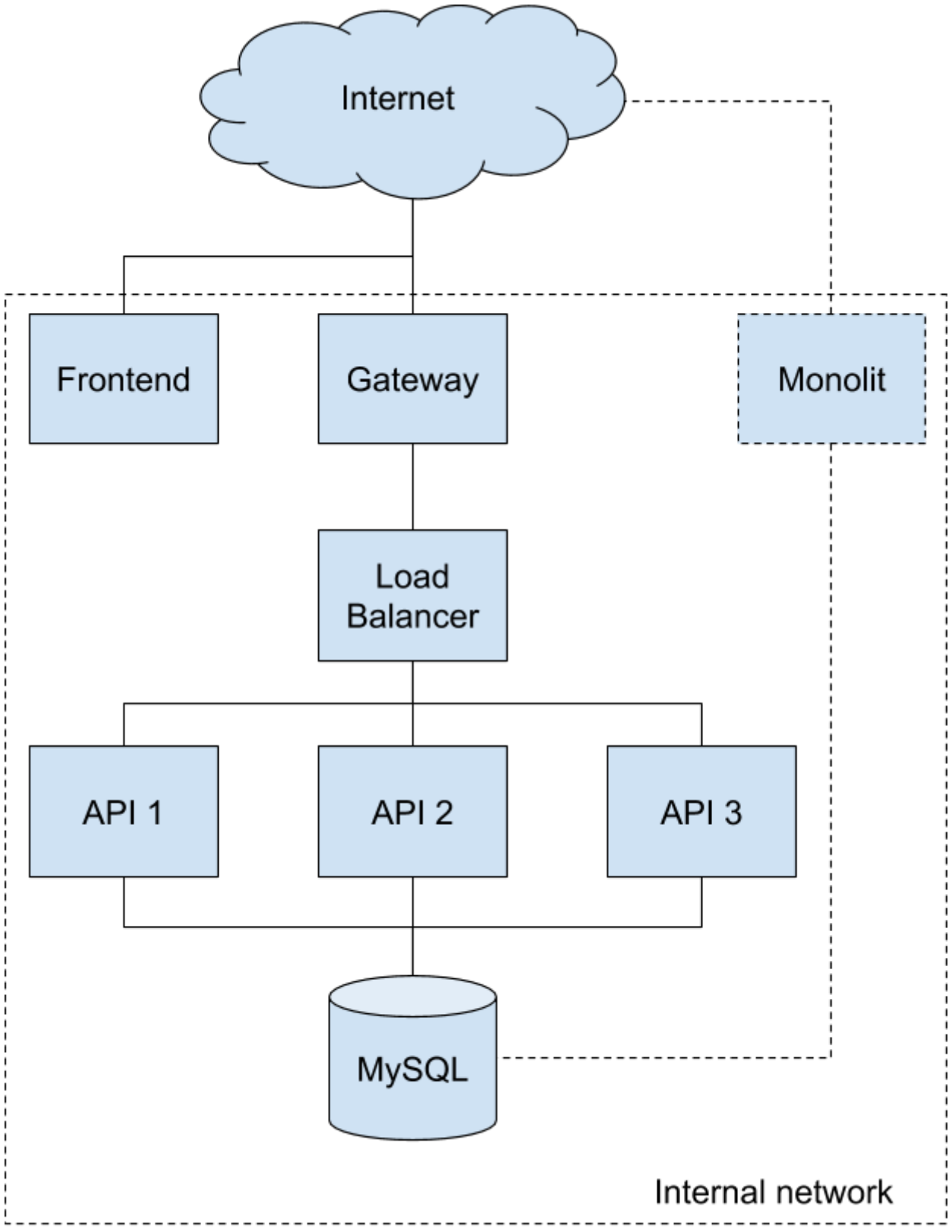
Architektura systemu:



Gateway: Akka HTTP (Scala) - filtrowanie ruchu, weryfikacja autoryzacji

Frontend: Apache HTTP + jQuery AJAX Client

Monolit: Apache HTTP + PHP 7.2.3 + Mysqli (UWAGA: monolit miewa problemy z dockerem i może nie działać, poprawię to przed kolejnymi zajęciami)

Load Balancer: HAProxy - balansuje ruchem, korzysta z algorytmu “Round robin”

API: Akka HTTP (Scala) + Slick + json4s (Jackson)

Adresacja:

Host	Adres (Linux)	Adres Windows/OSX
Frontend	10.10.0.200:80	docker_machine_ip:80
Monolit	10.10.0.200:80/monolith	docker_machine_ip:80/monolith
Gateway	10.10.0.10:8000	docker_machine_ip:8000
Load Balancer	10.10.0.100:9000	docker_machine_ip:9000
API 1	10.10.0.101:9000	docker_machine_ip:9010
API 2	10.10.0.102:9000	docker_machine_ip:9020
API 3	10.10.0.103:9000	docker_machine_ip:9030
MySQL	10.10.0.110:3306	docker_machine_ip:3306

Autentykacja i nagłówki:

Połączenie poprzez Gateway wymaga nagłówka “Authorization” ustawionego na wartość: “**94ab54b5-f120-4290-a4ec-587dee09206e**”. Połączenie bezpośrednio do serwerów API czy przez Load Balancer nie wymagają autoryzacji.

Połączenie do części endpointów wymaga nagłówka z tokenem sesji użytkownika (uzyskany przez zapytanie do /authenticate - patrz str. 5) zgodnym z żądanym id:

“Http-Auth-Token” : <user_token>

Pola domyślne i ukryte:

Część danych nadawana jest w sposób automatyczny - wszelkie numery ID encji, statusy zamówień, takie dane nie podlegają zmianie.

Część nadawana jest w sposób niejawny dla użytkownika - suma kontrolna hasła, czy token użytkownika. Te dane nie są zwracane wraz z resztą danych o użytkowniku.

Stronicowanie:

Zapytania zwracające listę encji zwracają domyślnie pierwsze 10 encji. Przyjmują jednak opcjonalny parametr url “page=<int>” który pozwala na pobranie kolejnych stron z wynikami.

API:

Jeśli nie jest zaznaczone inaczej system operuje na danych w formacie JSON.

Health check:

GET: /

Zwraca unikalne id serwera jako *application/json*

GET: /health

Zwraca unikalne id serwera jako *text/html*

Użytkownicy:

GET: /users

Pobiera listę użytkowników. Wykorzystuje stronicowanie.

POST: /users

Dodaje nowego użytkownika, wymagane pola:

- firstname
- lastname
- email (unikatowe)
- username (unikatowe)
- city
- address
- birthdate (np. 1990-01-01)
- password

GET: /users/<userId>

Pobiera dane użytkownika o id = userId. Wymaga tokena użytkownika.

POST, PUT: /users/<userId>

Aktualizuje dane użytkownika. Wymaga tokena użytkownika. Ignoruje zmianę pól:

- id
- email
- username
- password
- token

GET: /users/<userId>/wallet

Zwraca stan konta użytkownika. Wymaga tokenu użytkownika.

GET: /users/<userId>/wallet/payin?value=<amount>

Pozwala na wpłatę środków na konto użytkownika. Wymaga parametru “value” w url. Parametr ten przyjmuje wartość numeryczną całkowitą dodatnią. Wymaga tokena użytkownika.

GET: /users/<userId>/orders

Zwraca listę zamówień użytkownika. Wymaga tokena użytkownika i wykorzystuje stronicowanie.

POST: /users/<userId>/orders

Pozwala na dodanie nowego zamówienia do konta użytkownika. Wymaga tokena użytkownika. Wymagane pola:

- userid
- screeningid
- ticketscount

GET: /users/<userId>/orders/<orderId>

Zwraca dane na temat zamówienia o id=orderId. Wymaga tokena użytkownika.

GET: /users/<userId>/orders/<orderId>/pay

Pozwala na opłacenie zamówienia. Wymaga tokena użytkownika i odpowiednich środków w portfelu użytkownika.

Logowanie:

POST: /authenticate

Pozwala na zalogowanie użytkownika (wygenerowanie tokenu). Wymaga pól:

- username
- password

Zamówienia:

GET: /orders

Pozwala na pobranie listy wszystkich zamówień w systemie. Wykorzystuje stronicowanie.

GET: /orders/<orderId>

Pozwala na pobranie informacji o zamówieniu o id = orderId.

Filmy:

GET: /movies

Pobiera listę filmów w systemie. Wykorzystuje stronicowanie.

POST: /movies

Pozwala na dodanie nowego filmu do systemu. Wymagane pola:

- title
- director
- genre
- country
- year
- tags
- agelimit

GET: /movies/<movieId>

Pobiera dane filmu o id = movieId.

GET: /movies/<movieId>/screenings

Pobiera listę wszystkich seansów danego filmu. Wykorzystuje stronicowanie.

GET: /movies/search

Pozwala na wyszukiwanie filmów. Przyjmuje dwa parametry url z których przynajmniej jeden musi być podany:

- title - pozwala na wyszukanie po całym lub częściowym tytule
- genre - pozwala na wyszukanie po dokładnym dopasowaniu do gatunku

Wykorzystuje stronicowanie.

Seanse:

GET: /screenings/

Zwraca listę wszystkich seansów w systemie. Wykorzystuje stronicowanie.

POST: /screenings/

Pozwala na dodanie nowego seansu do systemu. Wymagane pola:

- movieid
- city
- roomnumber
- seatslimit
- seatstaken
- time
- price

Encje systemu:

Użytkownik (user):

- id : Int (nadawany automatycznie, unikatowy)
- firstname : String
- lastname : String
- email : String (unikatowy)
- username : String (unikatowy)
- city : String
- address: String
- birthdate: String (format YYYY-MM-DD)
- password: String (niejawny przy pobieraniu)
- token: String (niejawny przy pobieraniu, nadawany automatycznie)

Portfel użytkownika (wallet):

- id: Int (nadawany automatycznie, unikatowy)
- userId: Int (unikatowy)
- value: BigDecimal (lecz lepiej używać Int - problemy przy zapisie do bazy)

Logowanie (authenticate):

- username: String
- password: String

Token użytkownika (authenticateResponse):

- user: User (zawiera dane o użytkowniku)
- token: String

Film (movie):

- id: Int (nadawany automatycznie, unikatowy)
- title: String
- director: String
- genre: String
- country: String
- year: Int
- tags: String
- agelimit: Int

Zamówienie (order):

- id: Int (nadawany automatycznie, unikatowy)
- userid: Int
- screeningid: Int
- status : String (nadawany automatycznie)
- ticketscount : Int
- totalprice: BigDecimal (nadawany automatycznie)

Seans (screening):

- id: Int
- movieid: Int
- city: String
- roomnumber: Int
- seatslimit: Int
- seatstaken: Int
- time: String (format YYYY-MM-DD HH-MM-SS)
- price: BigDecimal