Here's a comprehensive list of all API calls with their expected inputs and outputs:

**Authentication API (authAPI)**

**1. Register**

javascript

authAPI.register(userData)

**Endpoint:** POST /api/auth/register **Sends:**

javascript

{

firstName: string,

lastName: string,

email: string,

password: string,

timezone: string

}

**Returns:**

javascript

{

success: true,

data: {

user: User,

token: string

},

message: string

}

**2. Login**

javascript

authAPI.login(email, password)

**Endpoint:** POST /api/auth/login **Sends:**

javascript

{

email: string,

password: string

}

**Returns:**

javascript

{

success: true,

data: {

user: User,

token: string

},

message: string

}

**3. Get Profile**

javascript

authAPI.getProfile()

**Endpoint:** GET /api/auth/profile **Sends:** Authorization header with Bearer token **Returns:**

javascript

{

success: true,

data: {

user: User

},

message: string

}

**4. Update Profile**

javascript

authAPI.updateProfile(userData)

**Endpoint:** PUT /api/auth/profile **Sends:**

javascript

{

firstName?: string,

lastName?: string,

email?: string,

timezone?: string

}

**Returns:**

javascript

{

success: true,

data: {

user: User

},

message: string

}

**5. Delete Profile**

javascript

authAPI.deleteAccount()

**Endpoint:** DELETE /api/auth/profile **Sends:** Authorization header **Returns:**

javascript

{

success: true,

message: string

}

**6. Change Password**

javascript

authAPI.changePassword(passwordData)

**Endpoint:** POST /api/auth/change-password **Sends:**

javascript

{

currentPassword: string,

newPassword: string

}

**Returns:**

javascript

{

success: true,

message: string

}

**Accounts API (accountsAPI)**

**1. Get User’s Accounts**

javascript

accountsAPI.getAll()

**Endpoint:** GET /api/accounts **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

accounts: Account[]

},

message: string

}

**2. Create Account**

javascript

accountsAPI.create(accountData)

**Endpoint:** POST /api/account **Sends:**

javascript

{

name: string,

type: "CHECKING" | "SAVINGS" | "CREDIT\_CARD" | "INVESTMENT",

balance?: number,

bankName?: string,

accountNumber?: string,

creditLimit?: number

}

**Returns:**

javascript

{

success: true,

data: {

account: Account

},

message: string

}

**4. Update Account**

javascript

accountsAPI.update(id, accountData)

**Endpoint:** PUT /api/account/{id} **Sends:**

javascript

{

name?: string,

type?: "CHECKING" | "SAVINGS" | "CREDIT\_CARD" | "INVESTMENT",

bankName?: string,

accountNumber?: string,

creditLimit?: number,

}

**Returns:**

javascript

{

success: true,

data: {

account: Account

},

message: string

}

**5. Delete Account**

javascript

accountsAPI.delete(id)

**Endpoint:** DELETE /api/account/{id} **Sends:** Authorization header **Returns:**

javascript

{

success: true,

message: string

}

**6. Get Account Balance**

javascript

accountsAPI.getBalance(id)

**Endpoint:** GET /api/accounts/{id}/balance **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

balance: number,

},

message: string

}

**7. Adjust Balance**

javascript

accountsAPI.adjustBalance(id, adjustment)

**Endpoint:** POST /api/accounts/{id}/balance/adjust **Sends:**

javascript

{

amount: number,

reason: string

}

**Returns:**

javascript

{

success: true,

data: {

account: Account

},

message: string

}

**8. Get Account Transactions**

javascript

accountsAPI.getTransactions(id)

**Endpoint:** GET /api/accounts/{id}/transactions **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

transactions: Transaction[]

},

message: string

}

**Transactions API (transactionsAPI)**

**1. Get All Transactions**

javascript

transactionsAPI.getAll(params?)

**Endpoint:** GET /api/transactions **Sends:** Optional query parameters **Query Params:**

javascript

{

limit?: number,

offset?: number,

page?: number,

accountId?: number,

categoryId?: number,

type?: "INCOME" | "EXPENSE" | "TRANSFER",

dateFrom?: string,

dateTo?: string

}

**Returns:**

javascript

{

success: true,

data: {

transactions: Transaction[],

pagination?: {

total: number,

page: number,

limit: number,

totalPages: number

}

},

message: string

}

**2. Create Transaction**

javascript

transactionsAPI.create(transactionData)

**Endpoint:** POST /api/transactions **Sends:**

javascript

{

amount: number,

description: string,

accountId: number,

categoryId: number,

type: "INCOME" | "EXPENSE" | "TRANSFER",

date?: string,

tags?: string[],

notes?: string

}

**Returns:**

javascript

{

success: true,

data: {

transaction: Transaction

},

message: string

}

**3. Get Transaction by ID**

javascript

transactionsAPI.getById(id)

**Endpoint:** GET /api/transactions/{id} **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

transaction: Transaction

},

message: string

}

**4. Update Transaction**

javascript

transactionsAPI.update(id, transactionData)

**Endpoint:** PUT /api/transactions/{id} **Sends:**

javascript

{

amount?: number,

description?: string,

categoryId?: number,

tags?: string[],

notes?: string

}

**Returns:**

javascript

{

success: true,

data: {

transaction: Transaction

},

message: string

}

**5. Delete Transaction**

javascript

transactionsAPI.delete(id)

**Endpoint:** DELETE /api/transactions/{id} **Sends:** Authorization header **Returns:**

javascript

{

success: true,

message: string

}

**6. Search Transactions**

javascript

transactionsAPI.search(params)

**Endpoint:** GET /api/transactions/search **Query Params:**

javascript

{

category?: string,

dateFrom?: string,

dateTo?: string,

amount?: number,

description?: string

}

**Returns:**

javascript

{

success: true,

data: {

transactions: Transaction[]

},

message: string

}

**7. Transfer Funds**

javascript

transactionsAPI.transfer(transferData)

**Endpoint:** POST /api/transaction/transfer **Sends:**

javascript

{

fromAccountId: number,

toAccountId: number,

amount: number,

description: string

}

**Returns:**

javascript

{

success: true,

data: {

transactions: Transaction[]

},

message: string

}

**8. Split Transaction**

javascript

transactionsAPI.split(splitData)

**Endpoint:** POST /api/transaction/split **Sends:**

javascript

{

originalTransactionId: number,

splits: [

{

amount: number,

categoryId: number,

description?: string

}

]

}

**Returns:**

javascript

{

success: true,

data: {

transactions: Transaction[]

},

message: string

}

**9. Get Pending Transactions**

javascript

transactionsAPI.getPending()

**Endpoint:** GET /api/transactions/pending **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

transactions: Transaction[]

},

message: string

}

**10. Approve Transaction**

javascript

transactionsAPI.approve(id)

**Endpoint:** PUT /api/transactions/{id}/approve **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

transaction: Transaction

},

message: string

}

**Categories API (categoriesAPI)**

**1. Get All Categories**

javascript

categoriesAPI.getAll()

**Endpoint:** GET /api/categories **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

categories: Category[]

},

message: string

}

**2. Create Category**

javascript

categoriesAPI.create(categoryData)

**Endpoint:** POST /api/category **Sends:**

javascript

{

name: string,

icon: string,

color: string,

type: "INCOME" | "EXPENSE",

parentId?: number,

description?: string

}

**Returns:**

javascript

{

success: true,

data: {

category: Category

},

message: string

}

**3. Get Category by ID**

javascript

categoriesAPI.getById(id)

**Endpoint:** GET /api/category/{id} **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

category: Category

},

message: string

}

**4. Update Category**

javascript

categoriesAPI.update(id, categoryData)

**Endpoint:** PUT /api/categories/{id} **Sends:**

javascript

{

name?: string,

icon?: string,

color?: string,

description?: string

}

**Returns:**

javascript

{

success: true,

data: {

category: Category

},

message: string

}

**5. Delete Category**

javascript

categoriesAPI.delete(id)

**Endpoint:** DELETE /api/categories/{id} **Sends:** Authorization header **Returns:**

javascript

{

success: true,

message: string

}

**7. Get Category Transactions**

javascript

categoriesAPI.getTransactions(id)

**Endpoint:** GET /api/categories/{id}/transactions **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

transactions: Transaction[]

},

message: string

}

**8. Get Category Spending Trend**

javascript

categoriesAPI.getSpendingTrend(id)

**Endpoint:** GET /api/category/{id}/spending-trend **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

trend: [

{

period: string,

amount: number

}

]

},

message: string

}

**Budgets API (budgetsAPI)**

**1. Get All Budgets**

javascript

budgetsAPI.getAll()

**Endpoint:** GET /api/budgets **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

budgets: Budget[]

},

message: string

}

**2. Create Budget**

javascript

budgetsAPI.create(budgetData)

**Endpoint:** POST /api/budgets **Sends:**

javascript

{

name: string,

amount: number,

period: "WEEKLY" | "MONTHLY" | "YEARLY",

categoryIds?: number[],

startDate?: string,

endDate?: string,

alertThreshold?: number

}

**Returns:**

javascript

{

success: true,

data: {

budget: Budget

},

message: string

}

**3. Get Budget by ID**

javascript

budgetsAPI.getById(id)

**Endpoint:** GET /api/budgets/{id} **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

budget: Budget

},

message: string

}

**4. Update Budget**

javascript

budgetsAPI.update(id, budgetData)

**Endpoint:** PUT /api/budgets/{id} **Sends:**

javascript

{

name?: string,

amount?: number,

period?: "WEEKLY" | "MONTHLY" | "YEARLY",

categoryIds?: number[],

alertThreshold?: number

}

**Returns:**

javascript

{

success: true,

data: {

budget: Budget

},

message: string

}

**5. Delete Budget**

javascript

budgetsAPI.delete(id)

**Endpoint:** DELETE /api/budgets/{id} **Sends:** Authorization header **Returns:**

javascript

{

success: true,

message: string

}

**6. Get Current Budgets**

javascript

budgetsAPI.getCurrent()

**Endpoint:** GET /api/budgets/current **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

budgets: Budget[]

},

message: string

}

**7. Get Budget Progress**

javascript

budgetsAPI.getProgress(id)

**Endpoint:** GET /api/budget/{id}/progress **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

progress: {

spent: number,

remaining: number,

percentage: number

}

},

message: string

}

**8. Get Budget Alerts**

javascript

budgetsAPI.getAlerts(id)

**Endpoint:** GET /api/budgets/{id}/alerts **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

alerts: Alert[]

},

message: string

}

**9. Reset Budget**

javascript

budgetsAPI.reset(id)

**Endpoint:** POST /api/budgets/{id}/reset **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

budget: Budget

},

message: string

}

**Goals API (goalsAPI)**

**1. Get All Goals**

javascript

goalsAPI.getAll()

**Endpoint:** GET /api/goals **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

goals: Goal[]

},

message: string

}

**2. Create Goal**

javascript

goalsAPI.create(goalData)

**Endpoint:** POST /api/goal **Sends:**

javascript

{

name: string,

description: string,

type: "SAVINGS" | "DEBT\_PAYOFF" | "INVESTMENT",

targetAmount: number,

currentAmount?: number,

targetDate?: string

}

**Returns:**

javascript

{

success: true,

data: {

goal: Goal

},

message: string

}

**3. Get Goal by ID**

javascript

goalsAPI.getById(id)

**Endpoint:** GET /api/goal/{id} **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

goal: Goal

},

message: string

}

**4. Update Goal**

javascript

goalsAPI.update(id, goalData)

**Endpoint:** PUT /api/goal/{id} **Sends:**

javascript

{

name?: string,

description?: string,

targetAmount?: number,

targetDate?: string

}

**Returns:**

javascript

{

success: true,

data: {

goal: Goal

},

message: string

}

**5. Delete Goal**

javascript

goalsAPI.delete(id)

**Endpoint:** DELETE /api/goals/{id} **Sends:** Authorization header **Returns:**

javascript

{

success: true,

message: string

}

**6. Get Goal Progress**

javascript

goalsAPI.getProgress(id)

**Endpoint:** GET /api/goal/{id}/progress **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

progress: {

percentage: number,

onTrack: boolean,

projectedCompletion?: string

}

},

message: string

}

**7. Add Goal Contribution**

javascript

goalsAPI.addContribution(id, contribution)

**Endpoint:** POST /api/goal/{id}/contribute **Sends:**

javascript

{

amount: number,

note?: string

}

**Returns:**

javascript

{

success: true,

data: {

goal: Goal

},

message: string

}

**8. Get Achieved Goals**

javascript

goalsAPI.getAchieved()

**Endpoint:** GET /api/goal/achieved **Sends:** Authorization header **Returns:**

javascript

{

success: true,

data: {

goals: Goal[]

},

message: string

}

**Authentication Headers**

All authenticated endpoints require:

javascript

headers: {

"Authorization": "Bearer {token}",

"Content-Type": "application/json"

}

For file uploads (CSV/Bank Statement import), the Content-Type is automatically set by the browser when using FormData.

**1. User Service**

**Dependencies:**

* spring-boot-starter-web
* spring-boot-starter-data-jpa
* spring-boot-starter-security
* spring-boot-starter-validation
* jjwt-api, jjwt-impl, jjwt-jackson
* spring-security-crypto
* postgresql/mysql driver
* spring-boot-starter-actuator
* spring-cloud-starter-netflix-eureka-client

**Database Schema:**

* Users table (id, email, password, firstName, lastName, timezone, createdAt)
* Roles table
* User preferences table

**Endpoints:**

* POST /auth/login
* POST /auth/register
* GET /auth/profile
* PUT /users/{id}
* POST /auth/refresh-token

**2. Account Service**

**Dependencies:**

* spring-boot-starter-web
* spring-boot-starter-data-jpa
* spring-boot-starter-security (for JWT verification)
* spring-boot-starter-validation
* postgresql/mysql driver
* spring-cloud-starter-netflix-eureka-client
* spring-cloud-starter-openfeign (to call User Service)

**Database Schema:**

* Accounts table (id, userId, name, type, balance, creditLimit, bankName, accountNumber, isActive)
* Account types enum

**Endpoints:**

* GET /accounts (user's accounts)
* POST /accounts
* PUT /accounts/{id}
* DELETE /accounts/{id}
* GET /accounts/{id}/balance

**3. Transaction Service**

**Dependencies:**

* spring-boot-starter-web
* spring-boot-starter-data-jpa
* spring-boot-starter-security
* spring-boot-starter-validation
* postgresql/mysql driver
* spring-cloud-starter-netflix-eureka-client
* spring-cloud-starter-openfeign (to call Account Service)
* spring-boot-starter-amqp (RabbitMQ for events)

**Database Schema:**

* Transactions table (id, accountId, amount, description, date, type, categoryId, pending)
* Categories table (id, name, icon, color)
* Transaction tags table

**4. Budget Service**

**Dependencies:**

* spring-boot-starter-web
* spring-boot-starter-data-jpa
* spring-boot-starter-security
* postgresql/mysql driver
* spring-cloud-starter-netflix-eureka-client
* spring-cloud-starter-openfeign (to call Transaction Service)
* spring-boot-starter-amqp

**Database Schema:**

* Budgets table (id, userId, name, amount, period, totalSpent, status)
* Budget categories table
* Budget alerts table

**5. Goal Service**

**Dependencies:**

* spring-boot-starter-web
* spring-boot-starter-data-jpa
* spring-boot-starter-security
* postgresql/mysql driver
* spring-cloud-starter-netflix-eureka-client
* spring-cloud-starter-openfeign

**Database Schema:**

* Goals table (id, userId, name, description, type, targetAmount, currentAmount, targetDate)
* Goal contributions table
* Goal milestones table

**8. API Gateway**

**Dependencies:**

* spring-cloud-starter-gateway
* spring-cloud-starter-netflix-eureka-client

**9. Service Discovery (Eureka Server)**

**Dependencies:**

* spring-cloud-starter-netflix-eureka-server
* spring-boot-starter-security (optional)

**Configuration:**

* Service registry
* Health checks
* Load balancing