**TDD for Operations Dashboard**

**Task**

1st Level requirement:

Architect, Design, Build E2E Solution for Operations Dashboard using .Net framework which captures metrics for Daily/Weekly/Monthly View comprising:

a) Shows top 5 applications with maximum number of recurring job failures

b) Shows top 5 applications with longest running jobs

c) Shows top 5 applications with maximum number of failures

d) Shows applications with P1/P2 incidents

2nd Level requirements:

Enhance the completed Operations Dashboard using .Net framework which captures metrics for Daily/Weekly/Monthly View. Add following enhancement:

Functional:

a) Shows top 5 applications with maximum number of recurring job failures -> On click of number of failures, open another table at the bottom with details of all those failures. There should be a keyword search option in this table.

b) Shows top 5 applications with longest running jobs -> Enhance the page to show a column with value of minimum threshold for each job which are running long

c) Shows top 5 applications with maximum number of failures -> On click on number of failures, it should open failure details with timestamp and other details. There should be a keyword search option in this table.

d) Shows top 5 applications with P1/P2 incidents -> Should show all P1/P2 incidents in table and open up details of incident on pop up.

**Analysis**:

**On requirement**:

Shows top 5 applications with maximum number of recurring job failures:

* Should show top 5 application name with those job which are failing continually.
* Should show all failure records with error log in separate table.

Shows top 5 applications with longest running jobs:

* Should show top 5 applications with the job which is taking longest running time as per the criteria (runtime greater than maximum runtime)

c) Shows top 5 applications with maximum number of failures:

* Should show top 5 those applications which has maximum count of job failures.
* Should show all failure records with error log in separate table.

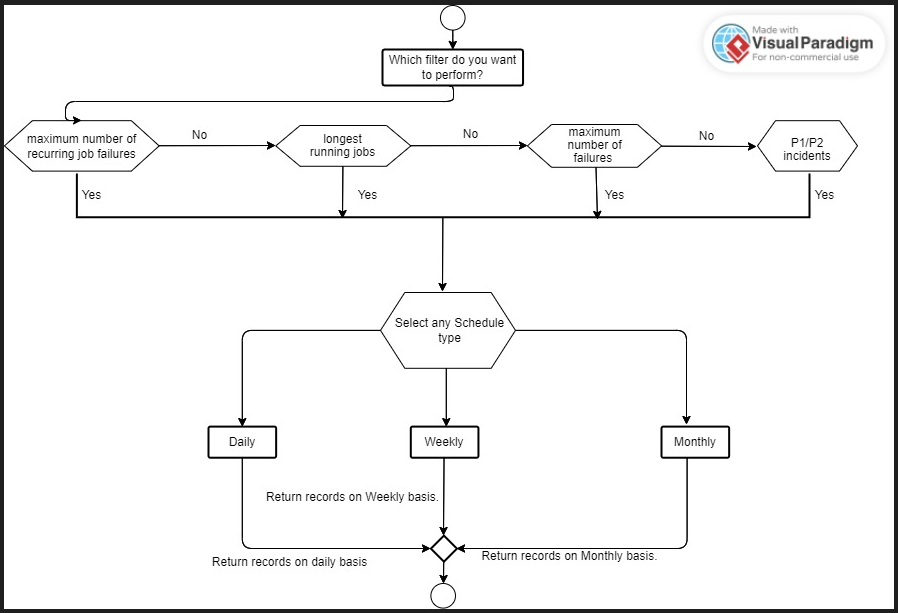
d) Shows top 5 applications with P1/P2 incidents:

* Should show all incidents which has P1/P2 priority application wise.
* Should open details of incident on pop up.

**Technologies**: ASP.Net MVC, ADO.Net, SQL Server, JavaScript, JQuery, AJAX, HTML, CSS, Bootstrap.

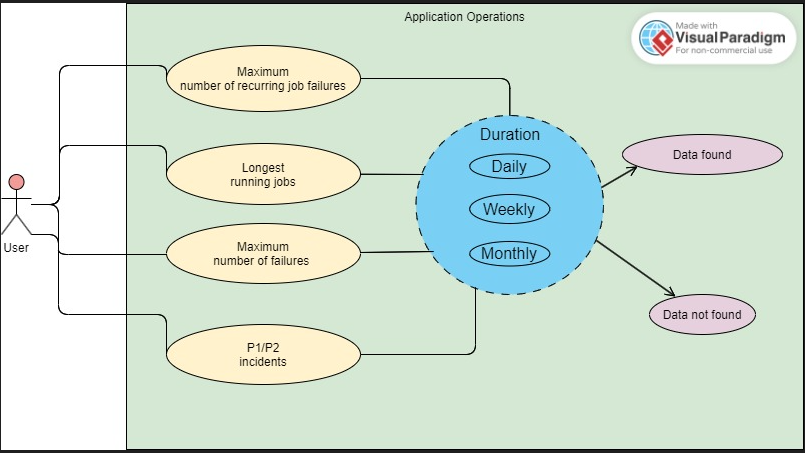
**Application Designs**:

1. Data Flow Diagram:



Data Flow diagram

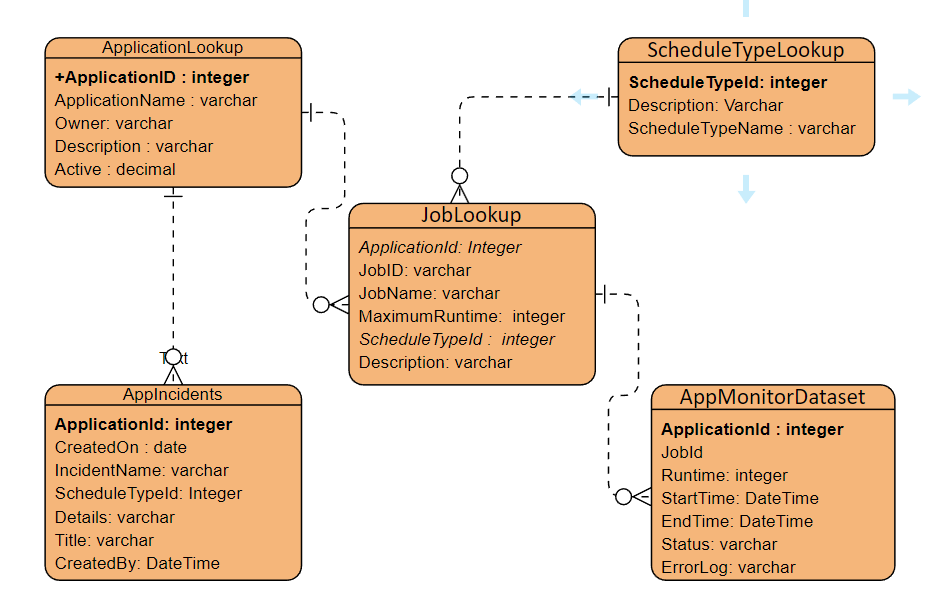
1. Use Case Diagram:



Use Case Diagram

**Database Design**:

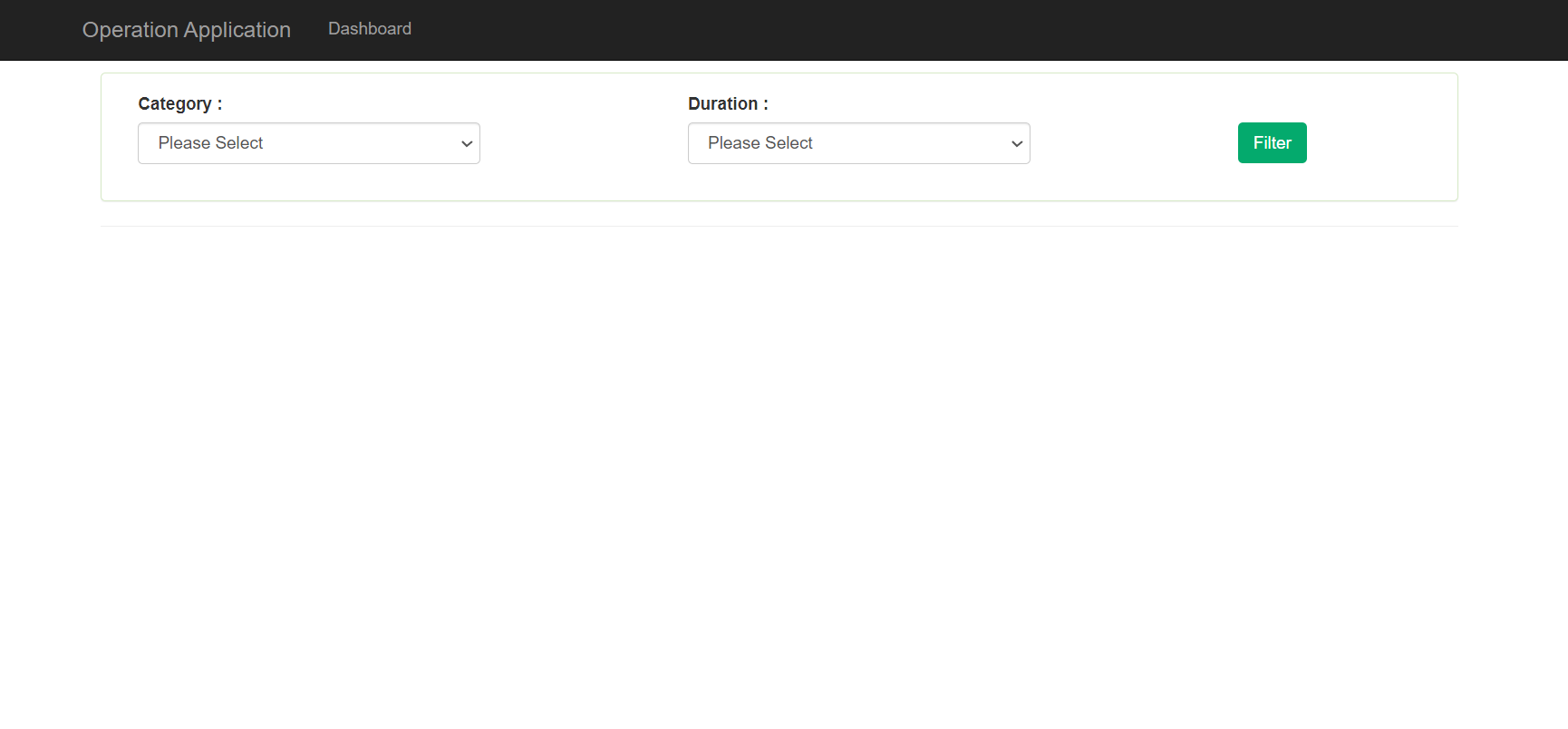
1. Entity Relationship Diagram:



|  |  |  |
| --- | --- | --- |
| Table Name | Table Columns | Table Description |
| ApplicationLookup | ApplicationId  ApplicationName  Owner  Description  Active | Application master table  contains details of application only |
| JobLookup | ApplicationId (PK of ApplicationLookup)  JobID  JobName  Description  MaximumRuntime : standard maximum runtime  ScheduleTypeId : contains job scheduler(Daily, Weekly, Monthly) | Job master table  contains details of job with maximum runtime and job scheduler with respect to application only |
| ScheduleTypeLookup | ScheduleTypeId : 1,2,3  Description  ScheduleTypeName : Daily  : Weekly  : Monthly | Scheduler master table  contains details of schedule duration as per application requirements only |
| AppMonitorDataset | ApplicationId  JobID  Runtime ->Time Taken by Job at run time  StartTime  EndTime  ErrorLog  Status -> 0 : Failure  1: Successful | Contain Job records for each Application with the time taken by Job at run time, status, and failure reason |
| AppIncidents | ApplicationId  IncidentNumber  CreatedOn  Priority -> P1, P2, P3, P4  ScheduleTypeId  Title  Details  CreatedBy | Contains all raised incidents on all application with priority and details. |

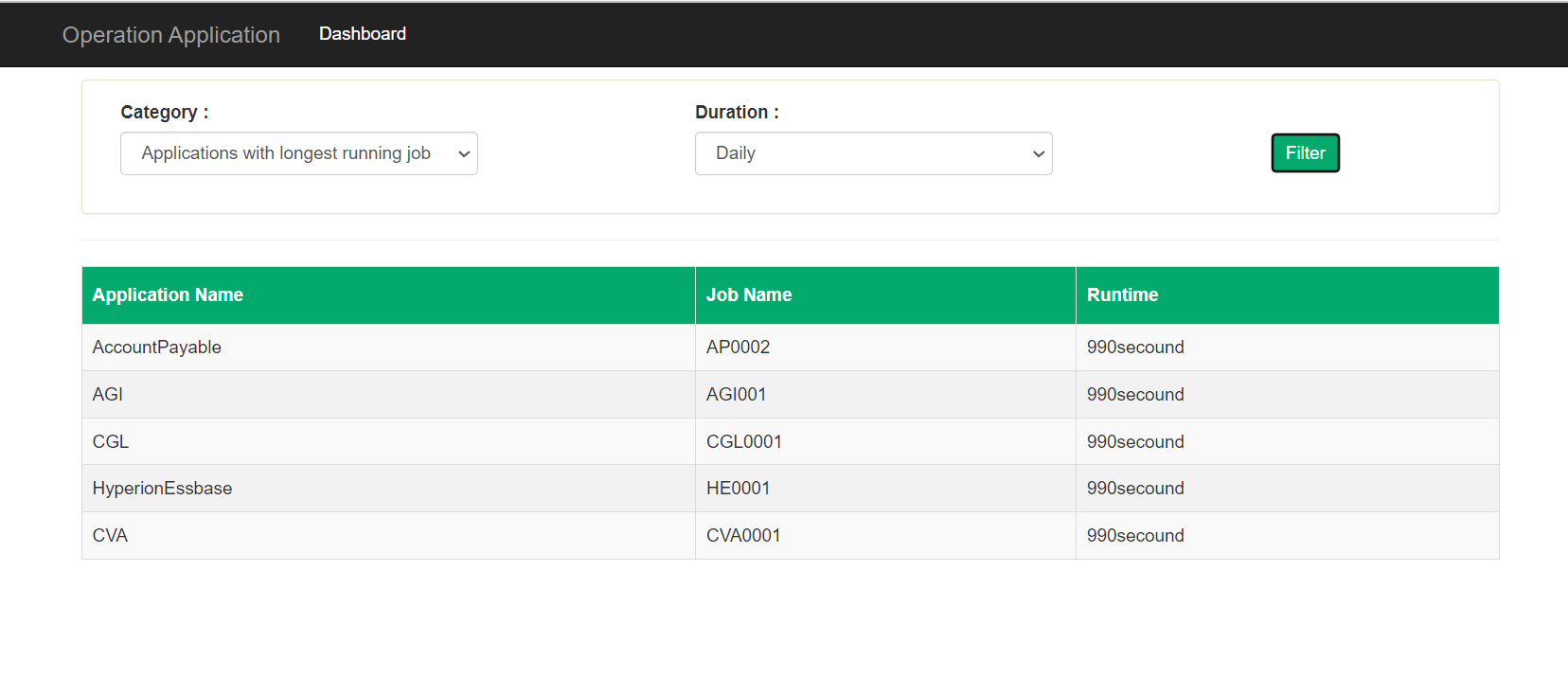
**UI Design**: It contains three HTML elements:

* 2 dropdowns (Category, Duration)
* Button (to hit event and fetch records as per dropdown selection)
* Table (to show requested record)

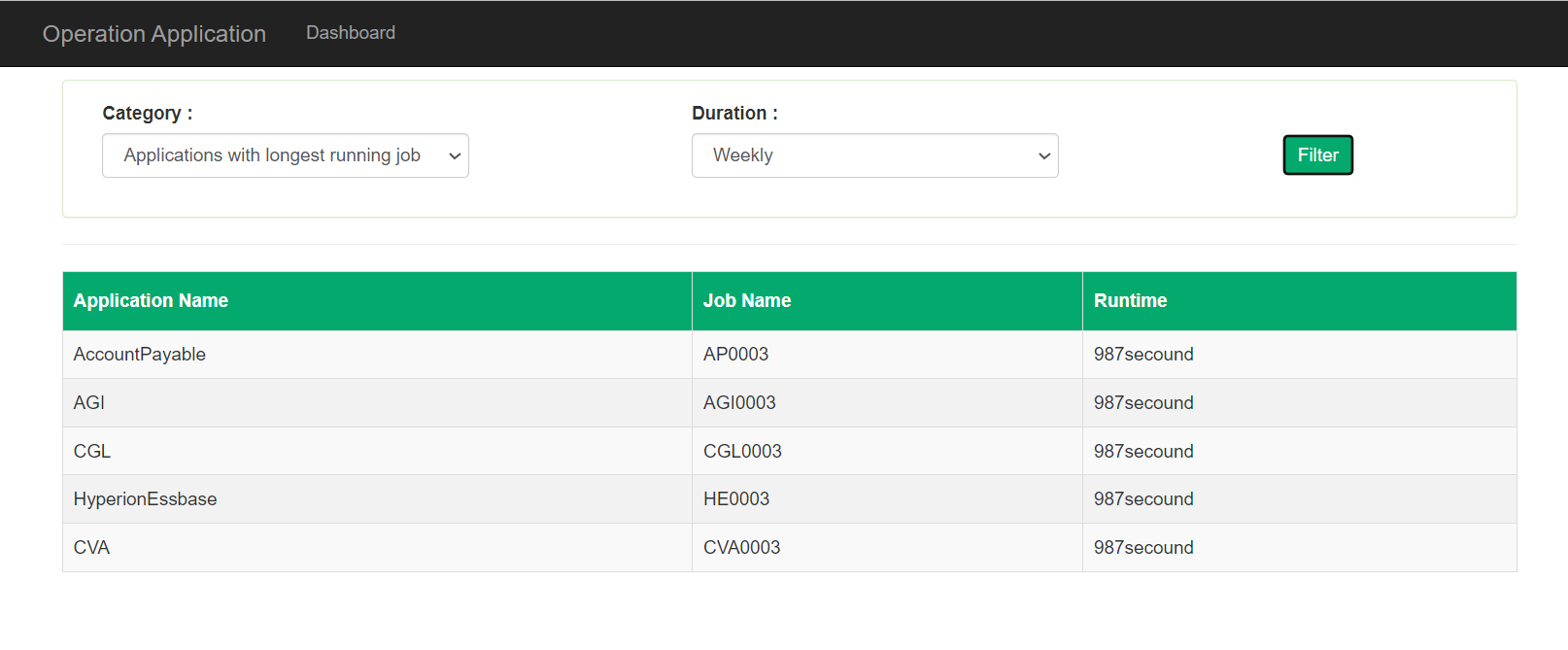


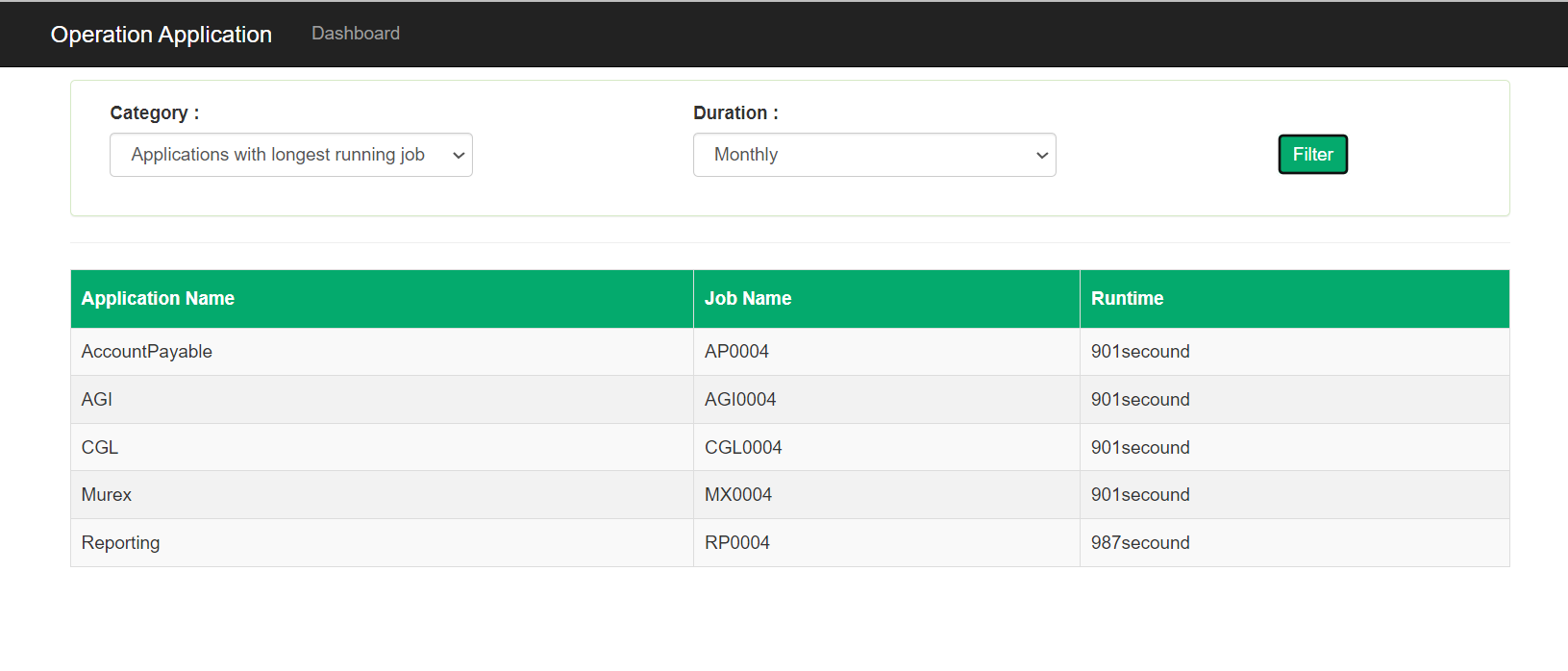
a) Shows top 5 applications with longest running jobs:

Daily basis:

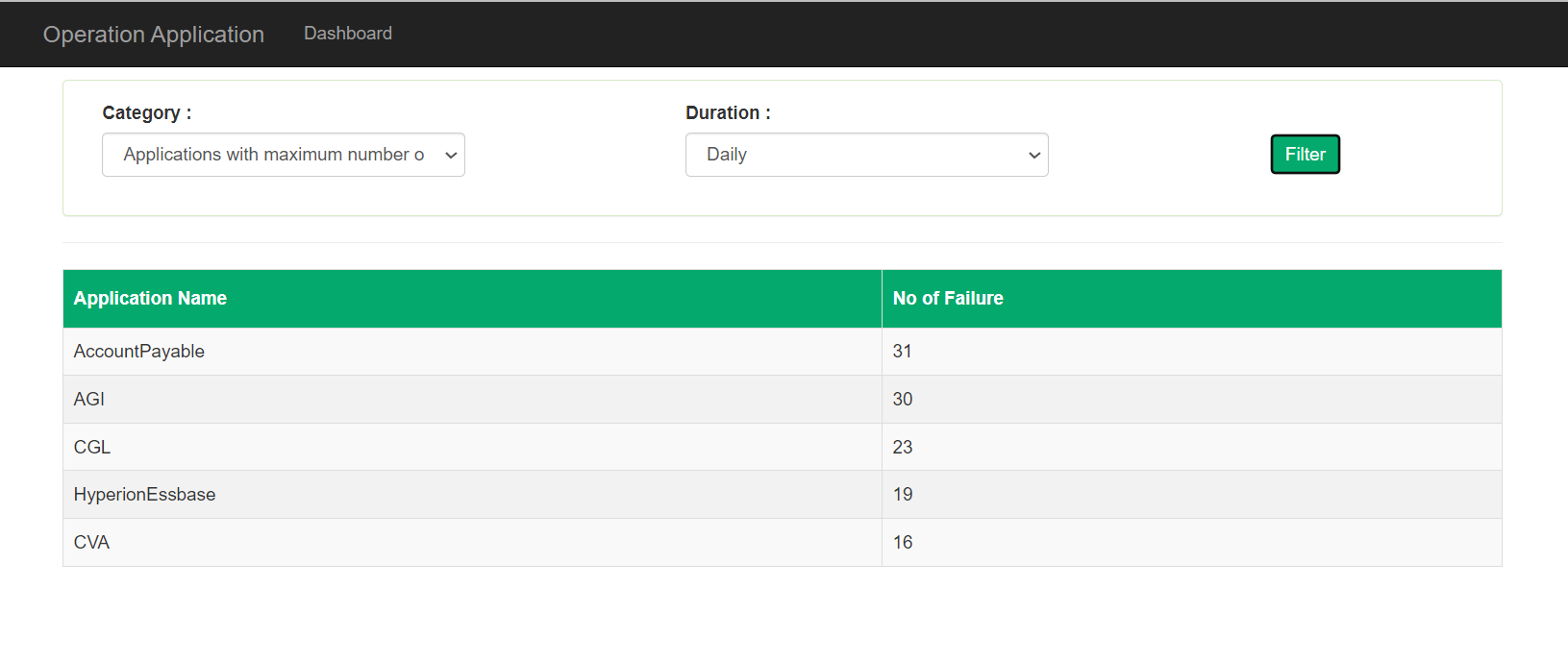


Weekly basis:

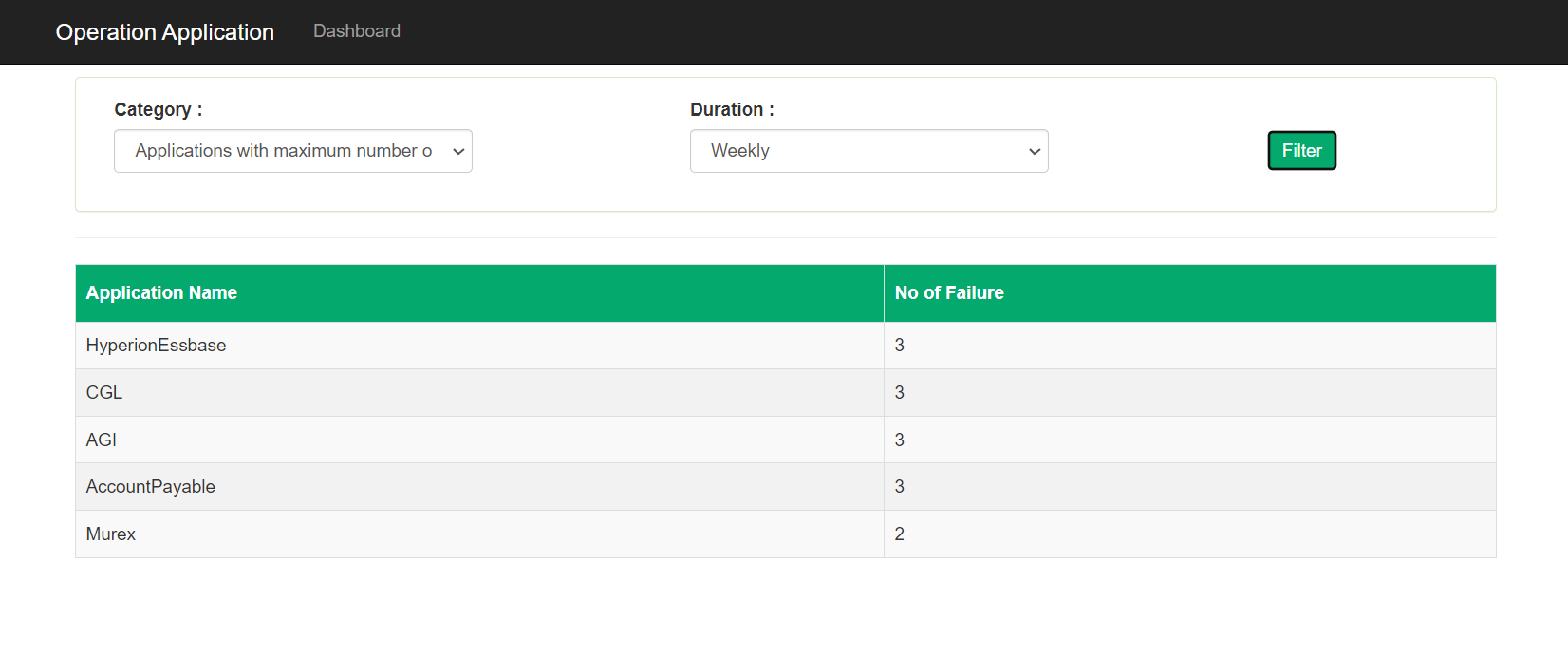


Monthly basis:

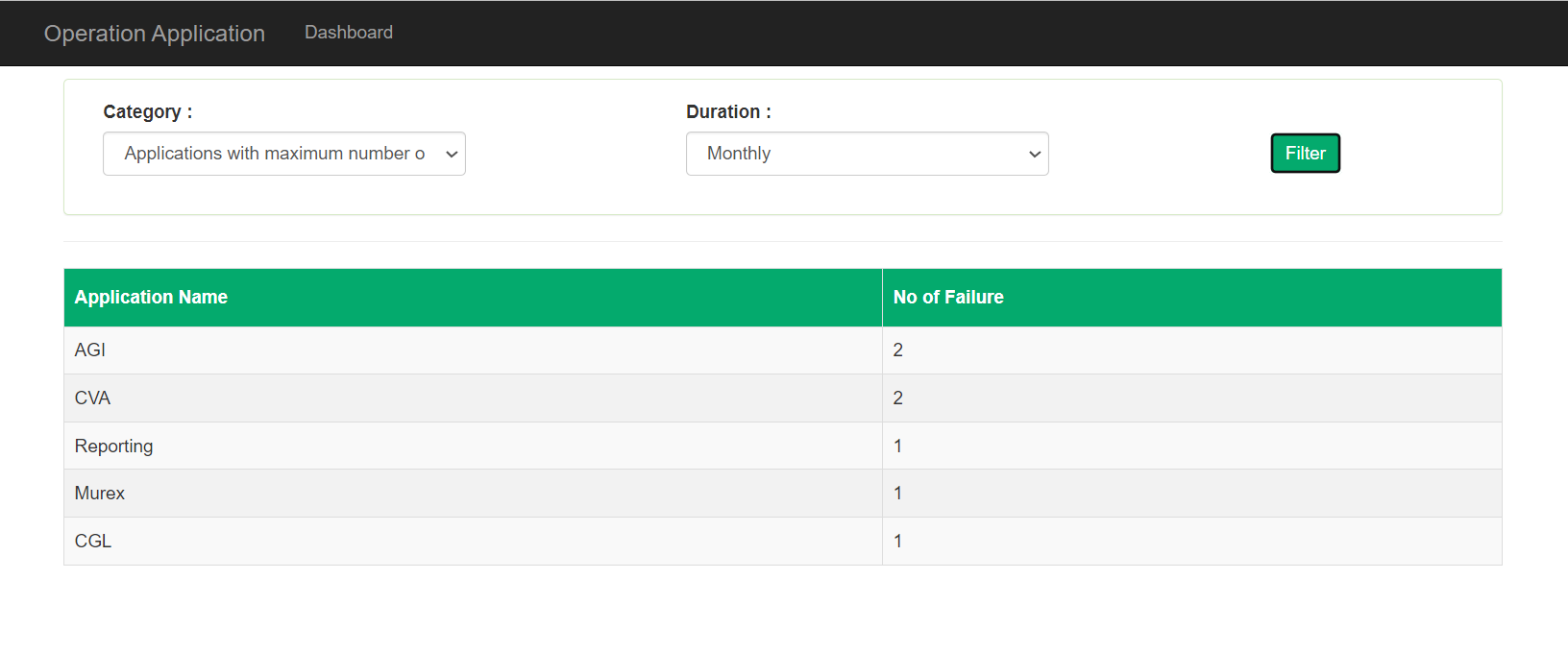
b) Shows top 5 applications with maximum number of failures:

Daily basis 

Weekly basis

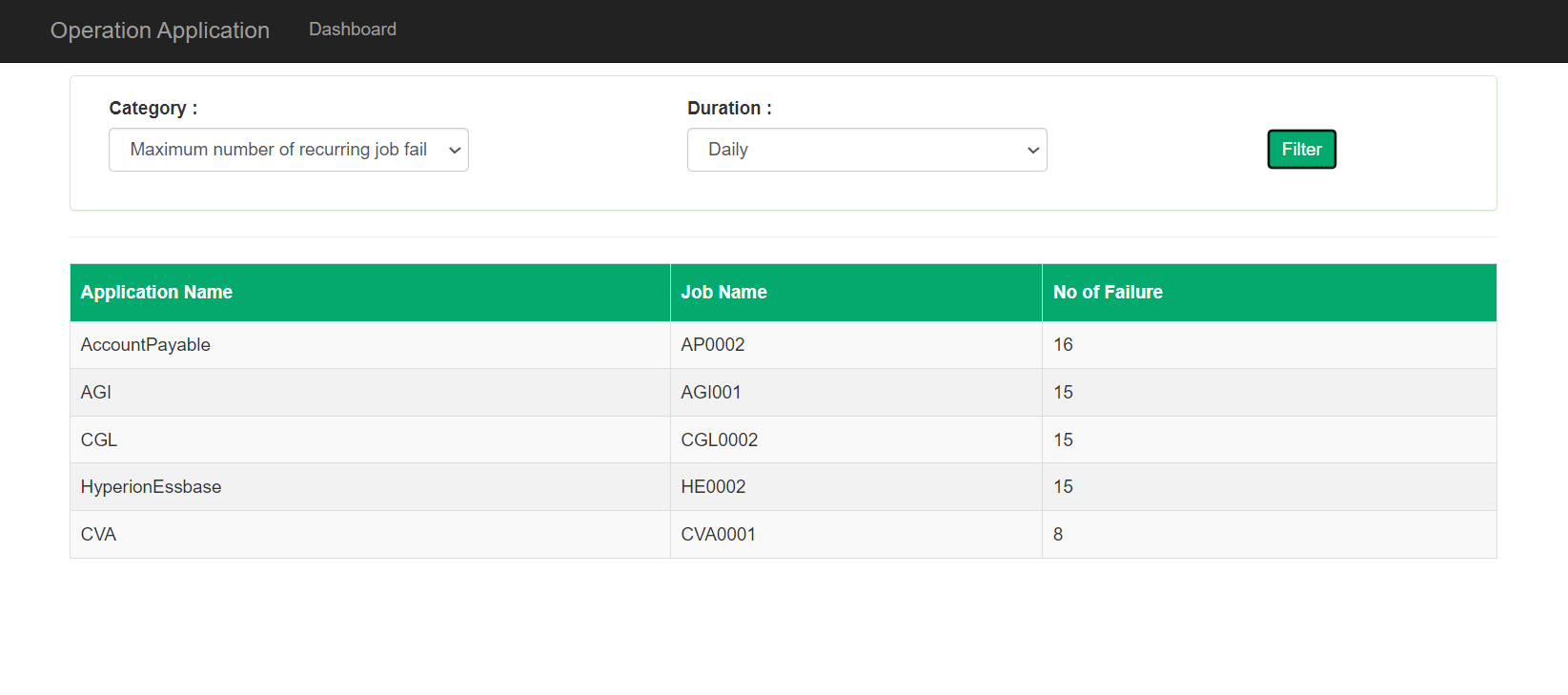


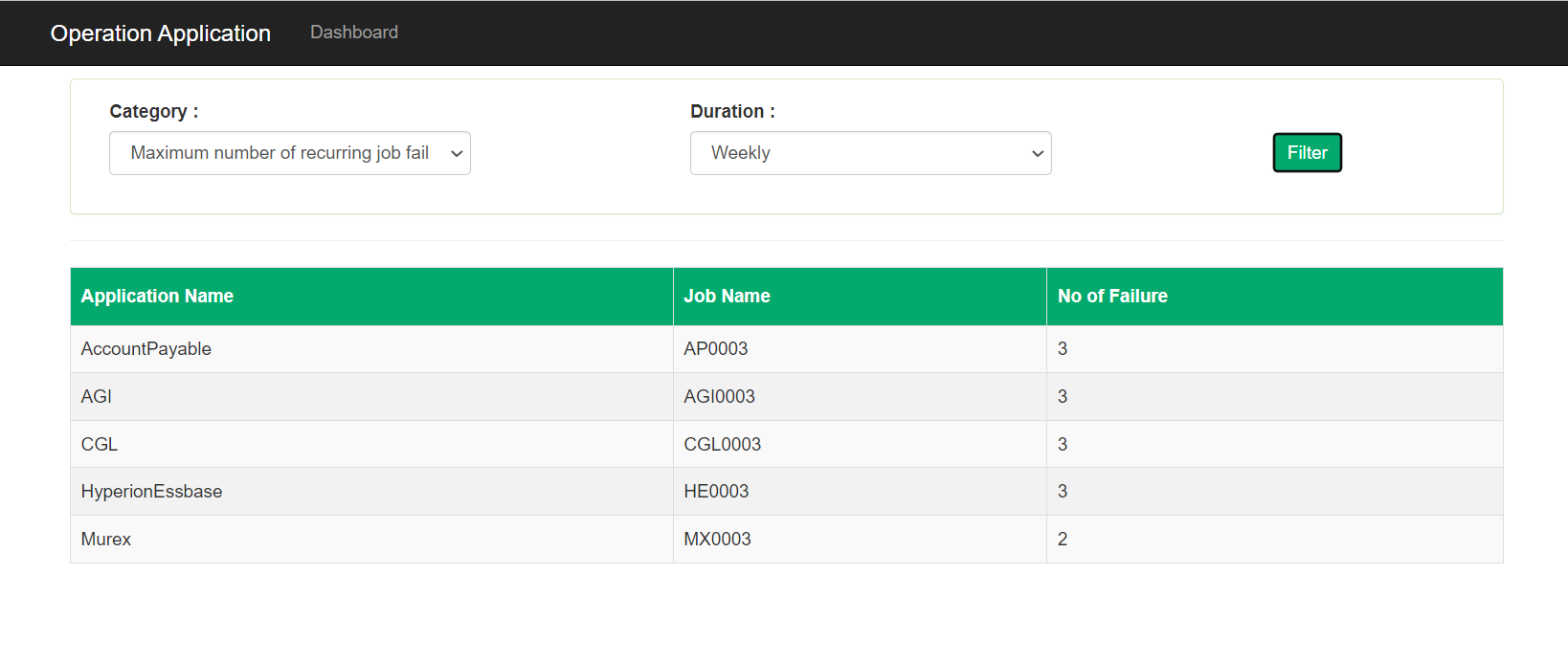
Monthly basis



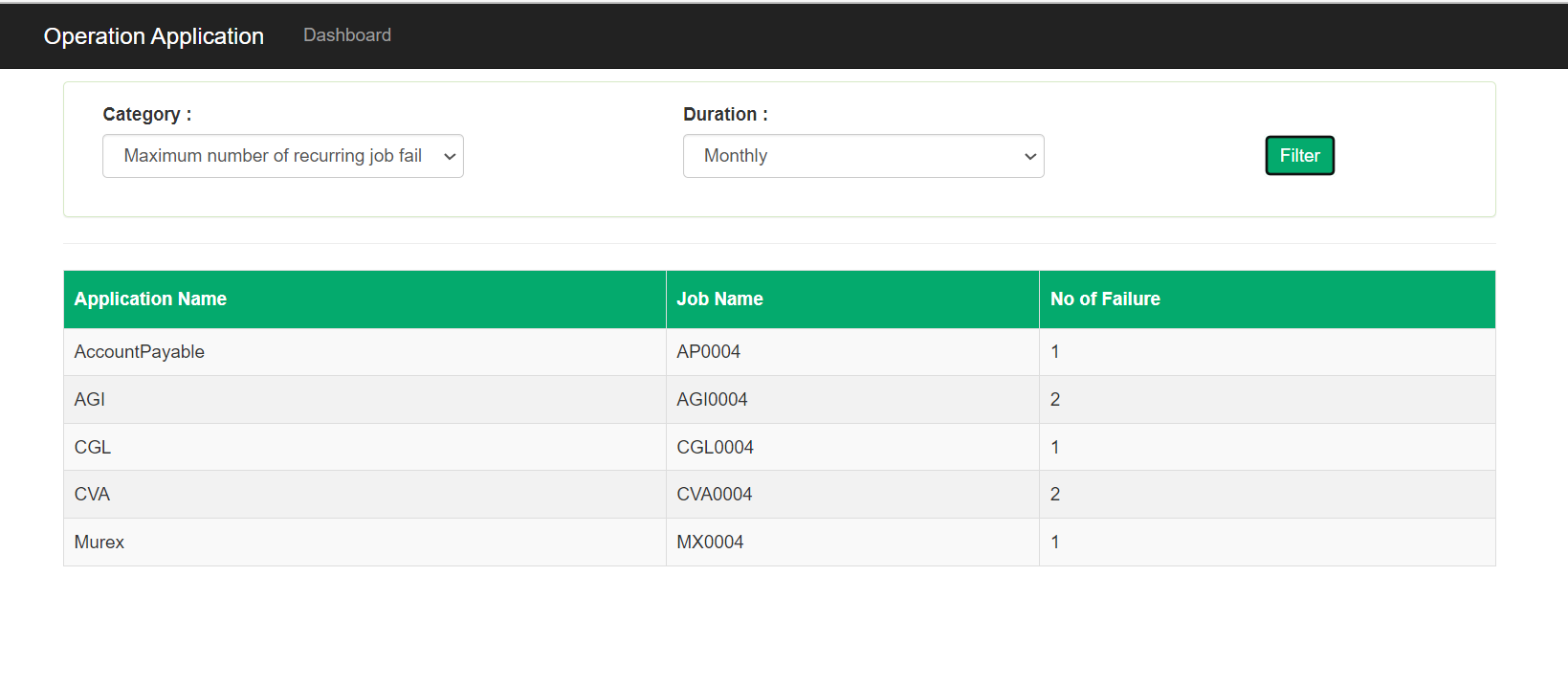
1. Shows top 5 applications with maximum number of recurring job failures

Daily basis



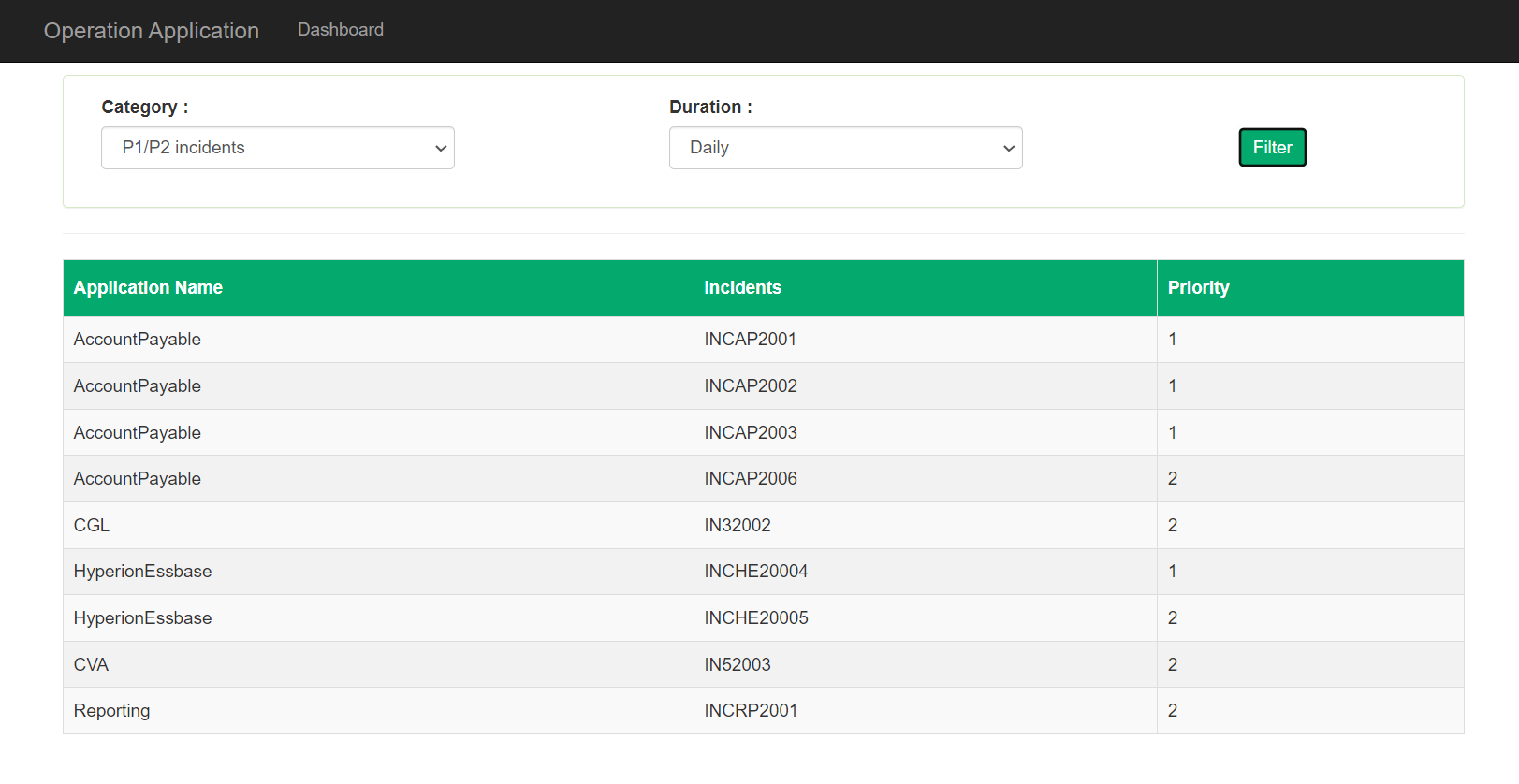
Weekly basis

Monthly basis

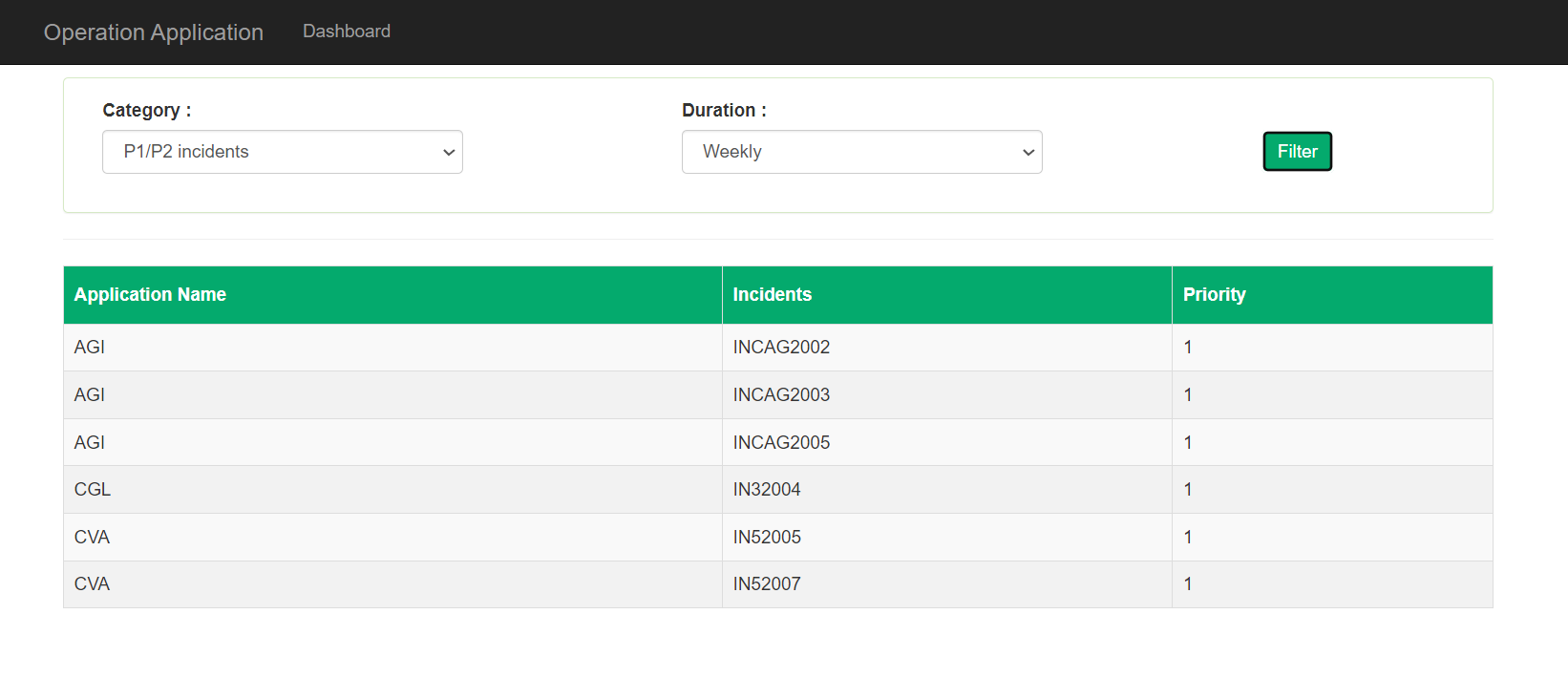


1. Shows top 5 applications with P1/P2 incidents

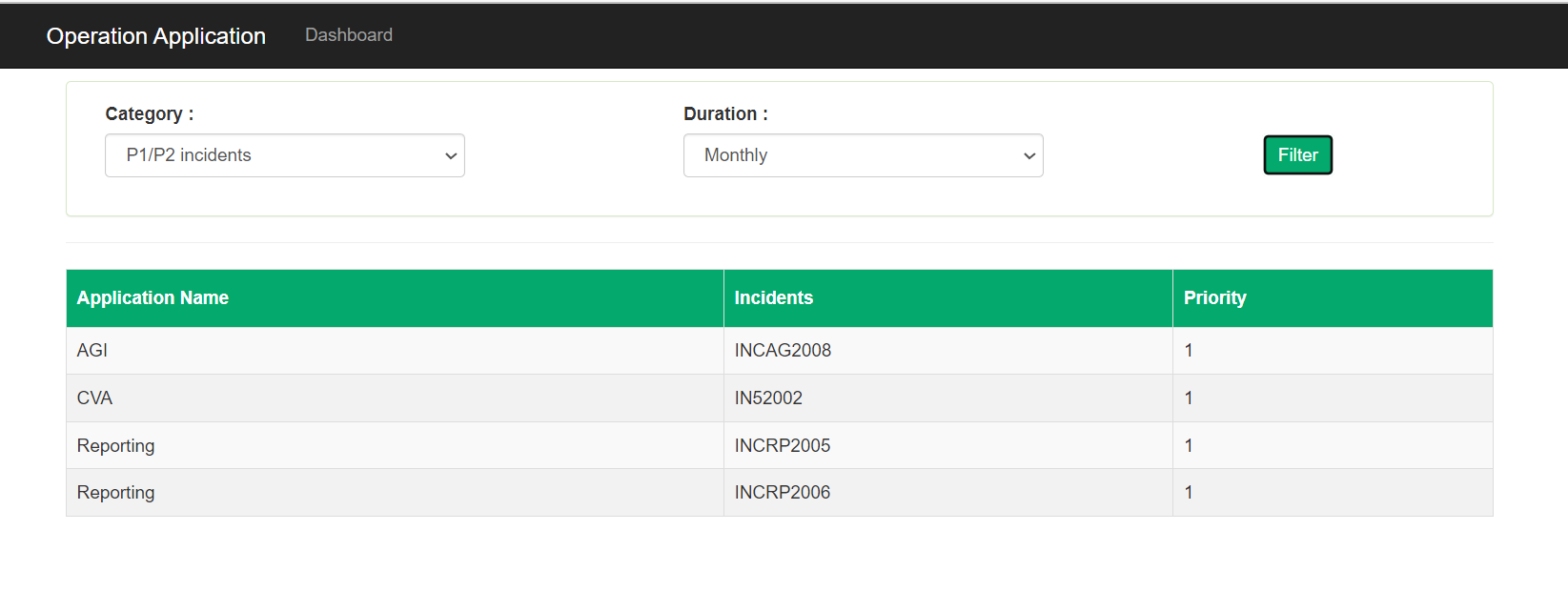
Daily basis



Weekly basis



Monthly basis



**Code** **Design**: To follow Single responsibility principle, at backcode code we are using 4-layer architecture.

**Database Layer**  
Responsibility: Set up connection with SQL server,   
Create query and hit into database,  
keeping records in model type after reading all records, return records to Business layer

**Business Layer**  
Responsibility: Get request from action method,   
Hit database layer for fetching expected records

**Controller**  
Responsibility: Get request from UI,  
Return response to UI

Model  
responsibility: Contain model view of SQL table and response architecture.

**Backend Code is following same path for each type of request. Those are the following steps:**

Show records with counts

1. API endpoint (<https://localhost:44364/Home/GetFilteredData?Duration=3&Category=4>)
2. Check API route and search for controller : “Home” then search for action method: “GetFilteredData” which can contain 2 int type of parameters.
3. Check if parameter has no value, return empty JSON to respective .cshtml page and show no records on UI.
4. If parameter has valid values, then controller calls to respective method in “Business Layer”– “HomeBusiness.cs”.
5. From “Business Layer”, It calls to “Database layer” to perform queries and get records - “SqlGenericRepository.cs”.
6. In “SqlGenericRepository.cs”., below steps were followed:
   1. Create SQL query
   2. Open database connection
   3. Execute command for fetching records from database
   4. Read values from reader
   5. Keep returned values in model
   6. Close the database connection.
   7. Return list of records from “SqlGenericRepository.cs” to “HomeBusiness.cs”.
7. In “HomeBusiness.cs”, it returns response to controller.
8. In “HomeController”, it returns response in JSON format to .cshtml page.

Job failure detailed records:

1. API endpoint ([https://localhost:44364/Home/GetJobData?id=”EF3009”&Duration=3](https://localhost:44364/Home/GetJobData?id=)
2. )
3. Check API route and search for controller : “Home” then search for action method: “GetJobData” which can contain 2 string type of parameters.
4. Check if parameter has no value, return empty JSON to respective .cshtml page and show no records on UI.
5. If parameter has valid values, then controller calls to respective method in “Business Layer”– “HomeBusiness.cs”.
6. From “Business Layer”, It calls to “Database layer” to perform queries and get records - “SqlGenericRepository.cs”.
7. In “SqlGenericRepository.cs”., below steps were followed:
   1. Create SQL query
   2. Open database connection
   3. Execute command for fetching records from database
   4. Read values from reader
   5. Keep returned values in model
   6. Close the database connection.
   7. Return list of records from “SqlGenericRepository.cs” to “HomeBusiness.cs”.
8. In “HomeBusiness.cs”, it returns response to controller.
9. In “HomeController”, it returns response in JSON format to .cshtml page.

Maximum Job failure detailed records with respect to:

1. API endpoint ([https://localhost:44364/Home/ GetApplicationJobData?id=”EF3009”&jobId=1&Duration=3](https://localhost:44364/Home/GetJobData?id=)
2. Check API route and search for controller : “Home” then search for action method: “GetApplicationJobData” which can contain 2 string type of parameters.
3. Check if parameter has no value, return empty JSON to respective .cshtml page and show no records on UI.
4. If parameter has valid values, then controller calls to respective method in “Business Layer”– “HomeBusiness.cs”.
5. From “Business Layer”, It calls to “Database layer” to perform queries and get records - “SqlGenericRepository.cs”.
6. In “SqlGenericRepository.cs”., below steps were followed:
   1. Create SQL query
   2. Open database connection
   3. Execute command for fetching records from database
   4. Read values from reader
   5. Keep returned values in model
   6. Close the database connection.
   7. Return list of records from “SqlGenericRepository.cs” to “HomeBusiness.cs”.
7. In “HomeBusiness.cs”, it returns response to controller.
8. In “HomeController”, it returns response in JSON format to .cshtml page.

Incident details:

1. API endpoint ([https://localhost:44364/Home/GetIncidentData?id=”INCAWE345”](https://localhost:44364/Home/GetIncidentData?id=)
2. Check API route and search for controller : “Home” then search for action method: “GetIncidentData” which can contain 2 string type of parameters.
3. Check if parameter has no value, return empty JSON to respective .cshtml page and show no records on UI.
4. If parameter has valid values, then controller calls to respective method in “Business Layer”– “HomeBusiness.cs”.
5. From “Business Layer”, It calls to “Database layer” to perform queries and get records - “SqlGenericRepository.cs”.
6. In “SqlGenericRepository.cs”., below steps were followed:
   1. Create SQL query
   2. Open database connection
   3. Execute command for fetching records from database
   4. Read values from reader
   5. Keep returned values in model
   6. Close the database connection.
   7. Return list of records from “SqlGenericRepository.cs” to “HomeBusiness.cs”.
7. In “HomeBusiness.cs”, it returns response to controller.
8. In “HomeController”, it returns response in JSON format to .cshtml page.

**Frontend code is following same path for each type of request. Those are the following steps:**

1. Show index.cshtml page.
2. Get “Category” and “Duration” values and hit on Filter button.
3. If there is no selection for any one(category/duration), it shows alert with “Please select any valid input” message.
4. If selection is correct so AJAX will hit the API endpoint (https://localhost:44364/Home/GetFilteredData?Duration=<duration\_value>&Category=<category\_value>)
5. After getting response from backend, check if it returns empty JSON, message will show “No records found”.
6. If response has records so before populating new records, it will remove all old data from the table and will populate new records in the table as per category selection.

**Test scenarios:**

UI Testing:

1. if user does not select any option from “category” and “Duration”, alert will show with message as “*Please select valid inputs*”.
2. if user select any option for “Duration” but does not select any option from “category” only, alert will show with message as “*Please select valid inputs*”.
3. if user select any option for “category” but does not select any option from “duration” only, alert will show with message as “*Please select valid inputs*”.
4. if user selects valid option from “category” and “Duration”:
5. if records are getting in response, should follow below steps:

* Remove old records from UI.
* Another table should not be appeared.
* Show requested records in respective table.

1. If records are not getting in response, should show “*No records found*”.

Backend Testing:

1. If API endpoint is not valid, will not serve dashboard and will through error.
2. If API endpoint is valid, check requested inputs with null and empty string. If yes, so return empty JSON.
3. Tested on “Early code exit” pattern, which restrict to run whole code if mandatory conditions are not satisfying.
4. Tested database connection is closed after fetching records from database. Use close connection statement in “Finally” block.
5. Tested on each try-catch block for handling exceptions.

References:

[Checking link (office.net)](https://statics.teams.cdn.office.net/evergreen-assets/safelinks/1/atp-safelinks.html)

[DataTables example - HTML (DOM) sourced data](https://datatables.net/release-datatables/examples/data_sources/dom.html)

[ERD Example - ATM | Visual Paradigm Online (visual-paradigm.com)](https://online.visual-paradigm.com/app/diagrams/#diagram:proj=0&type=ERDiagram&gallery=/repository/7eb24322-267a-4239-8e0c-b8339b1cd46e.xml&name=ERD%20Example%20-%20ATM)

[operationdashboard / appmonitoring — Bitbucket](https://bitbucket.org/operationdashboard/appmonitoring/src/master/) [id; dikshagoyal1541998/ Bitbucket@1541998]