# Milestone 4

Java.Lang.NoClassFoundException

By: Caleb Post and Matthew Lash

# 2. Change History:

Revision Milestone 4

* Added API for the Photo Application
* Added Feature Listing and updated work assignments
* Added Architectural Evaluation and Improvement Section

Revision Milestone 3

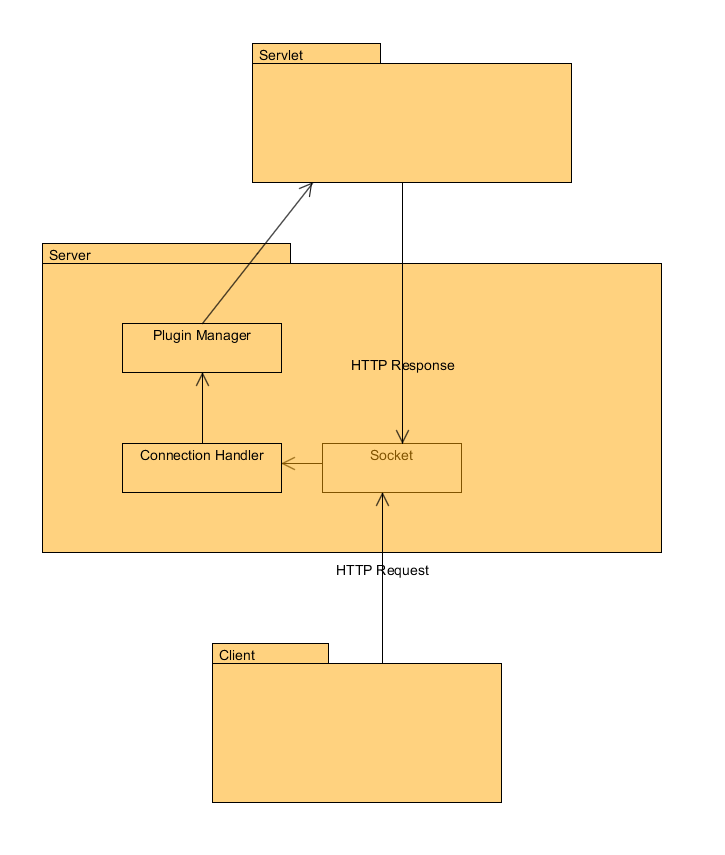
* Modified UML Diagram to reflect new methods and Logging System
* Added Tactics
* Added Improvements and Scenarios

Revision Milestone 2:

* Modified architecture to follow Servlets and Plugin Architecture
* Updated Architecture and UML Diagrams
* Added testing section for Servlet Features
* Added explanation of extension mechanism
* Feature Listing
* Expanded improvement section

# 3. Architecture and Design

The Architecture remained unchanged in this milestone. The entire application was written as a plugin that could be added to our server, meaning that the server architecture could remain unchanged.



UML Diagram Follows

# 4. Tactics/Feature Listing

## Milestone 4

Photo Plugin and Associated Servlets – Caleb

Get Servlet

Post Servlet

Delete Servlet

Put Servlet

Querying Servlet

Javascript and Client application – Matt

Ajax calls

HTML Integration

## Milestone 3

Logging and Audits – Caleb

Dos Attack Prevention – Connection Throttling – Matt

Caching of requested Files – Caleb

Memory Limiting and Refactoring – Matt

## Milestone 2

GET Requests: Design is was done by Caleb and Matt. Implementation was completed by Caleb.

POST Requests: Design is was done by Caleb and Matt. Implementation was completed by Caleb.

PUT Requests: Design is was done by Caleb and Matt. Implementation was completed by Caleb.

DELETE Requests: Design is was done by Caleb and Matt. Implementation was completed by Caleb.

Dynamic Loading: Design is was done by Caleb and Matt. Implementation was completed by Matt.

Root Context and Configurable Route: Design is was done by Caleb and Matt. Implementation was completed by Matt.

## Milestone 1

GET Requests: Refactoring done by Caleb

POST Requests: Implementation done by Caleb

PUT Requests: Implementation done by Matt

DELETE Requests: Implementation by Matt

# 5. Architectural Evaluation and Improvements

No additional improvements were made to availability, performance, or security in this milestone.

All the listings below are from the milestone 3 document.

## Availability

### Scenario 1

Source: Malicious User

Stimulus: Dos Attack

Artifact: Sever

Environment: Normal run condition

Response: It should cancel the connections that the attacker has

Response measure: How long server can survive before crashing

#### Test Plan

We first use the dos client to attack the server to see how long the server can continue to serve the other clients. Then we implement the improvement and see how long until the server stops server connections.

#### Before Improvement

Server seems to survive indefinitely without crashing, however while performing a SYN flood, it takes 13.12 seconds of time before the server starts rejecting connections periodically. 1000 requests per second.

It takes around 19 seconds for the system to start rejecting a significant number of connections.

#### Improvement tactics

After a threshold number of connections we ban the user from being able to make new connections and cancel the current connections. This threshold is based upon number of current connections, we allow each user to have a maximum number of concurrent connections. If a user exceeds this number of concurrent connections they will be blacklisted for a time period.

#### After Improvement

Server still seems to run indefinitely without crashing. It ran for 3:30 minutes until we stopped the test. It did not drop a significant number of connections as we saw in the initial testing. We stopped the test because we were seeing periodic connection drops which could be expected of a server, with no indication that the server would enter the state at which it would start dropping most connections.

### Scenario 2

Source: Users wanting to make requests

Stimulus: A user makes a request

Artifact: Server

Environment: Normal run conditions

Response: Server handles requests in parallel

Response measure: Total memory consumed by application

#### Test Plan

Tactic is to increase the capacity number of threads that can be running simultaneously on the server. This means limiting the memory impact of each individual thread. Implement the server such that it never needs to have entire files in memory or read entire contents of incoming requests before starting to service the connection. Increasing the total memory of the server could also be used for this tactic.

#### Before Improvement

During a DOS style attack, 18 threads run with an allocated memory of ~160MB on the heap of which it peaks to 40MB used.

Without a DOS attack and requesting a single large file uses a single thread which used a maximum of 3 MB change from the server when not requesting anything. (Difference from 33MB to 36MB) when requesting a video file of 118MB.

#### Improvement tactics

Tactic is to increase the capacity number of threads that can be running simultaneously on the server. This means limiting the memory impact of each individual thread. Implement the server such that it never needs to have entire files in memory or read entire contents of incoming requests before starting to service the connection. Increasing the total memory of the server could also be used for this tactic.

#### After Improvement

During a DOS attack: 20 Threads, increase of 2 threads in use, with allocated memory of 160MB same as before, with 48MB used at peak with cache implementation, dropping down to 6mb to 8mb used when the connections are dropped because of the DOS attack connection refusal.

Without the DOS attack: 3MB change when not requesting anything. (Difference from 12MB to 15MB)

## Performance

### Scenario 1

Source: A user who wants a file

Stimulus: A user requests a file

Artifact: Sever

Environment: Normal run condition

Response: The server serves the file

Response measure: The time between request and serving the content

#### Test Plan

We generate back to back requests of similar files and measure the time the server takes to responds. Then we implement our cashing technique and send the requests again and check to see if we get a better overall average response time.

#### Before Improvement

Initial Connection between 5ms and 7ms

Average connection latency after 100 requests of index page: 0.48623853211009177ms

Average connection latency after 30 request while DOS is hit: 11.072162087149598ms

#### Improvement tactics

This tactic is to implement a caching service for the server. For the most commonly requested resources, it should be in the cache, in memory, instead of needing to be read from disk with each request.

#### After Improvement

Without DOS: Initial Connection After: 7ms. Average Connection after improvement: 0.25ms

With DOS: 0.43902439024390244ms with latency dropping the longer we serviced. (Average latency, highest latency is loading the cache and is only done every cache clear.)

### Scenario 2

Source: A malicious user

Stimulus: A dos attack on our server

Artifact: Sever

Environment: stressed run condition

Response: The server stops the dos attack to allow other users to not get slower performance

Response measure: Latency for legitimate client under dos attack

#### Test Plan

We first use the dos client to attack the server to see how long the server can continue to serve the other clients. Then we implement the improvement and see how long until the server stops server connections.

#### Before Improvement

Server seems to survive indefinitely without crashing, however while performing a SYN flood, it takes 13.12 seconds of time before the server starts rejecting connections periodically. 1000 requests per second.

It takes around 19 seconds for the system to start rejecting a significant number of connections.

#### Improvement tactics

After a threshold number of connections we ban the user from being able to make new connections and cancel the current connections. This threshold is based upon number of current connections, we allow each user to have a maximum number of concurrent connections. If a user exceeds this number of concurrent connections they will be blacklisted for a time period.

#### After Improvement

During a DOS attack: 20 Threads, increase of 2 threads in use, with allocated memory of 160MB same as before, with 48MB used at peak with cache implementation, dropping down to 6mb to 8mb used when the connections are dropped because of the DOS attack connection refusal.

Without the DOS attack: 3MB change when not requesting anything. (Difference from 12MB to 15MB)

## Security

### Scenario 1

Source: A malicious user

Stimulus: A dos attack on our server

Artifact: Server

Environment: Normal run condition

Response: The server stops the dos attack to allow other users to not get slower performance

Response measure: How long the survive before crashing

#### Test Plan

We first use the dos client to attack the server to see how long the server can continue to serve the other clients. Then we implement the improvement and see how long until the server stops server connections.

#### Before Improvement

Server seems to survive indefinitely without crashing, however while performing a SYN flood, it takes 13.12 seconds of time before the server starts rejecting connections periodically. 1000 requests per second.

It takes around 19 seconds for the system to start rejecting a significant number of connections.

#### Improvement tactics

After a threshold number of connections we ban the user from being able to make new connections and cancel the current connections. This threshold is based upon number of current connections, we allow each user to have a maximum number of concurrent connections. If a user exceeds this number of concurrent connections they will be blacklisted for a time period.

#### After Improvement

During a DOS attack: 20 Threads, increase of 2 threads in use, with allocated memory of 160MB same as before, with 48MB used at peak with cache implementation, dropping down to 6mb to 8mb used when the connections are dropped because of the DOS attack connection refusal.

Without the DOS attack: 3MB change when not requesting anything. (Difference from 12MB to 15MB)

### Scenario 2

Source: A malicious user

Stimulus: User uploads virus

Artifact: Sever

Environment: Normal run condition

Response: The server should serve the request

Response measure: The server maintains an audit trail so the managers can find the malicious user

#### Test Plan

Tester 1 does some actions and tester 2 has to tell what different actions the first tester did. If he can successful do this for a variety of request then our audit system works. (he can send requests from different ip addresses)

#### Before Improvement

There is no way to backtrack malicious users

#### Improvement tactics

In order to track and trace the actions of potentially malicious individuals, maintain a log of all connections, requests, and origins such that they can be looked up at a later date to determine the actions of any user on the server. This will be implemented as a log file.

#### After Improvement

There is a way to backtrack malicious users

# 6. Application

## 6.1 About

The application is a storage system for photos. Users can create photos, pull photos, and request information about photos through the provided api. It allows them to remotely store their photos in an easily shareable fashion.

## 6.2 API

### 6.2.1 F1 Getting Picture Information

Description: Allows querying about whether a file exists with the specified filename. Will return code 200 if the file exists. Will return code 404 if the file does not exist.

Method: GET

URI: /photo/get/exists/<filename>

Response Body:

{

“code”: 200 or 404

}

Development Status: COMPLETE

### 6.2.1 F1 Getting Picture Information Cont.

Description: Allows getting the image specified by the filename requested. Returns the image file encoded as a base 64 string. Returns 404 if the image is not found.

Method: GET

URI: /photo/get/return/<filename>

Response Body Success:

{

“code”:200,

“image”:”<IMAGE\_DATA in Base 64>”

}

Response Body Failure:

{

“code”:404

}

Development Status: COMPLETE

### 6.2.2 F2 Creating a new Picture

Description: Allows the client to upload a new image to the server. Image must be encoded as a base 64 string. Returns back 200 if the image was successfully uploaded. Returns 500 if the server had an error while processing. Returns 403 if an image already exists with that filename.

Method: POST

URI: /photo/create

Request Body:

{

“filename”:”<FILE\_NAME>”,

“image”:”<IMAGE\_DATA in Base64>”

}

Response Body:

{

“status”:200 or 500 or 403

}

Development Status: COMPLETE

### 6.2.3 F4 Deleting a Picture

Description: Allows the client to delete a previously uploaded image. Returns 200 if the image with that name was deleted. Returns 500 for internal server error while attempting to delete. Returns 404 if the image with the filename requested was not found on the server.

Method: DELETE

URI: /photo/delete/<filename>

Response Body:

{

“status”: 200 or 500 or 404

}

Development Status: COMPLETE

### 6.2.4 F3 Editing a Picture

Description: Allows the client to replace the contents of a previously uploaded image. Returns 200 on successful upload. Returns 500 on error. Returns 404 if no file with that name was found to edit.

Method: PUT

URI: /photo/edit

Request Body:

{

“filename”:”<FILE\_NAME>”,

“image”:”<IMAGE\_DATA in Base64>”

}

Response Body:

{

“status”: 200 or 500 or 404

}

Development Status: COMPLETE

### 6.2.5 F5 Querying about Images

Description: Returns a list of filenames which match the criteria of the search term using it as a prefix for the files on the server. If no search term is specified, then all files on the server will be returned.

Method: GET

URI: /photo/list/<search-term>

Response Body:

{

“files”: [

“filename 1”,

“filename 2”,

…

],

“count”: 0 (number of files returned)

}

Development Status: COMPLETE

### 6.2.5 F5 Deleting Matched Images

Description: Deletes all photos on the server which match the criteria. If a file extension is specified it will delete all photos with that file extension. If no extension is specified, then all photos will be removed from the server.

Method: DELETE

URI: /photo/list/<image-extension>

Response Body:

{

“status”:200 or 500

“count”: 0 (count of the files deleted)

}

Development Status: COMPLETE

# 7. Future Improvements

## Milestone 4

Application could be made more secure by adding in an authentication layer which would require that the user have a valid token in order to make use of the photo saving, and getting abilities.

Application could be extended to allow multiple user accounts instead of having a global space in which all users upload their photos. Then each user account could have their own section of the photos that they are allowed to see and no one else.

## Milestone 3

Security could be improved further by implementing HTTPS by securing a certificate for the server and adding in the encryption layer.

Security could be improved further by authentication of users and requiring credentials for use.

Security could be improved further by implementing access controls based on served content.

Caching could be further improved to remove specific elements from cache as they are out of date instead of clearing the entire cache periodically. Logic could be added to cache only those files which are commonly requested.

## Milestone 2

Throttling mechanisms could be implemented to allow for the server to monitor connections that are consuming too many resources and limit them. This could be done proactively by blocking requests that are too consuming, or it could be done actively by killing threads that are consuming too many resources and blocking other clients.

Recovery mechanisms could be put into place such as request queueing and blocking if the server becomes stressed under too many requests being made at once.

If resources become a problem we may need to implement servlet cloning. Currently there is a single servlet instance hosted which handles all requests forwarded to that servlet. This works in the small scale scenario that we have been testing. We may need to multithread these servlets so that the same servlet can handle more than one request at a time, or we many need to clone the servlet in order to make each thread have a copy of it.

Preventative measures could be implemented such as monitoring connections and requests, blocking any users who appear to be malicious and consuming more connections. This would help with preventing DDOS attacks against the server.

## Milestone 1

As suggested by Chandan, we should do some work with a command pattern perhaps with the generation of the responses. The way that the code currently is, all the work is done within the factory and the system is not very extensible with the responses.

Also, we could encapsulate the handling the HTTP messages into a more plugin like architecture. This would make it so that we could also handle different types of requests beyond HTTP requests without extensive modification of the system, only through adding a new plugin which respects the design we have put in place.