# NextGen 4.0

## 1. Introduction

NextGen 4.0 is a new framework written from scratch on top of Spring Boot framework. This framework helps developer to focus mainly on business logic leaving aside the boilerplate code to be handled by the framework. This framework has been built from the perspective of developer-first mentality, also have abstracted many functionalities and have given out of box implementation.

The goal of this document is to help developer build new project from scratch by using different features of the framework. The developer should go through this document in its entirety before starting to use it.

## 2. Project Structure

*Recommended file hierarchy*

```
ng-demo/ ①
    ng-demo-web/ ②
    ng-demo-client/ ③
settings.gradle
build.gradle
gradlew.bat
gradlew
```

① **Root Project** - This the root project which can have multiple sub module depending on project requirement.

② **Web Module** - This contains Resource API and all the business logic

③ **Client Module** - This module will have Resource API constants, Request/Response POJOs, enums and other utility class. Also, this project will be published so that other project can reuse the files.

**TIP** Create sub modules as required for your project and abstract out files in those modules. Also publish the modules only if you think it can be used by other projects.

**NOTE** Brainstorm on project name and come up with a meaningful and self-explanatory name.

*Recommend file hierarchy for Web Project*

```
ng-demo-web/ ①
    src/
        main/
            java/
                com.deloitte.nextgen.demo ② ③
                    resources/ ④
                    services/ ⑤
                        impl/
                            DemoServiceImpl.java
                        DemoService.java
                    repositories/ ⑥
                        impl/
                            DemoRepositoryImpl.java
                        DemoRepository.java
                    entities/ ⑦
                    mappers/ ⑧
                    dto/ ⑨
                DemoApplicationSvc.java ⑩
            resources/ ⑪
                application.yml
        test/
            java/
build.gradle
```

① **Root package name** - Here the root package name is *com.deloitte.nextgen*

② **Sub package** - "demo" is the sub package, this can be name of the project. E.g dc (i.e. data collection), ar (i.e. application registration)

③ **Resources** - This package will have all the controller and api endpoints.

④ **Services** - This package will have all business logic classes.

⑤ **Repositories** - This package will have logic related to fetch data from databases.

⑥ **Entities** - This package will have all entities class.

⑦ **Mappers** - This package will have all mapper interfaces.

⑧ **Start Class** - This is spring boot starter file from where you application start.

⑨ **Data Transfer Objects (DTO)** - Create DTOs which can be used to transfer objects within you application

⑩ **Resources directory** - All the files related to configuration or properties will be placed in this directory.

*Recommend file hierarchy for Client Project*

```
ng-demo-client/ ①
    src/
        main/
            java/
                com.deloitte.nextgen.demo.client ② ③
                request/ ④
                response/ ⑤
                enums/ ⑥
                util/ ⑦
            resources/
                application.yml
        test/
            java/
build.gradle
```

① **Root package name** - Here the root package name is *com.deloitte.nextgen.demo.client*

② **Sub package** - "client" is the sub package.

③ **Request** - This package will contain all the request related POJO files.

④ **Response** - This package will contain all the response related POJO files.

⑤ **Enum** - This package will contain all the enums used in the project.

⑥ **Util** - This package will contain all utility classes.

⑦ **Resources directory** - All the files related to configuration or properties will be place in this directory.

**TIP** Create sub packages as needed for better segregation. This will help organize your files and have better readability.

**NOTE** Create only those file in client project which can be re-used by other project.

# 3. Getting Started

## 3.1. SDK, IDE and Software

- Download Java 8 from here. Choose Windows X64 product for JDK.
- Download IntelliJ Community Edition from here
- Download Visual Studio Community Edition from here
- Download Docker from here

**NOTE** All the above files are .exe files, double-clicking on the respective file will open installation wizard. Follow the steps in wizard and install it. We recommend installing Java 8 first and then other software.

## 3.2. Creating Project

The section will go through on how we can create a project and sub modules to get started. Please follow the step as mention below.

### 3.2.1. Root Project

1. Go to `File` > `New Project`
2. Select `Gradle` and click on `Next`

[Step 1] | *../images/project_step_1.PNG*

1. Mention the project name and mention the Group Id i.e. com.deloitte.nextgen.<**project_name**>
   e.g. com.deloitte.nextgen.**demo**. Click `Finish`.

[Step 2] | *../images/project_step_2.PNG*

### 3.2.2. Sub Module - Web

1. `Right Click` on Root Project Select `New` > `Modules`
2. Mention the module name and change the group id to com.deloitte.nextgen.<**project_name**>.<**module**>
   e.g. com.deloitte.nextgen.**demo**.**web** Click `Finish`.

[Step 3] | *../images/project_step_3.PNG*

### 3.2.3. Sub Module - Client

1. `Right Click` on Root Project Select `New` > `Modules`
2. Mention the module name and change the group id to com.deloitte.nextgen.<**project_name**>.<**module**>
   e.g. com.deloitte.nextgen.**demo**.**client** Click `Finish`.

[Step 4] | *../images/project_step_4.PNG*

### 3.2.4. Final Project layout

Below is the final project structure.

[Step 4] | *../images/final_project_struct.PNG*

**TIP** Repeat the above steps to create new modules.

**NOTE** Create base packages as per the naming convention section. Refer to Coding Guidelines.

[star] ***Congratulations, you are ready to write some awesome code now.*** [star]

### 3.3. Gradle

Each gradle project or module have build.gradle file which helps gradle to identify project related settings. We can add gradle plugins which eases the way we run/build project, create custom tasks, add dependency and many other project related changes. In this section we will be updating build.gradle files to update our project. To get more knowledge about gradle refer below quick links.

- Gradle User Guide
- Medium Article
- Tutorials Point
- Petri Kainulainen Blog

*Adding NG Framework Starter Dependency in Web Project*

```
dependencies {
    implementation 'com.deloitte.nextgen.framework:ng-spring-boot-starter:x.y.z'
}
```

*Adding NG Framework Independent module in a project*

```
dependencies {
    implementation 'com.deloitte.nextgen.framework:ng-web:x.y.z'
    implementation 'com.deloitte.nextgen.framework:ng-persistence:x.y.z'
}
```

**NOTE** Avoid adding starter dependency in your client project or any other module you create, since it won't require all dependency mentioned in starter project.

**TIP** Only add required dependency, avoid/remove unwanted dependencies. This will help in reduce size of jar.

# 4. Coding Guidelines

## 4.1. Files

| Type | Description | Examples |
|---|---|---|
| Package | The package name should start with **com.deloitte.nextgen** followed by project name or abbreviation if name is too long. | • com.deloitte.nextgen.**dc** — Abbreviation for data collection<br>• com.deloitte.nextgen.**correspondence** — Self explanatory<br>• com.deloitte.nextgen.**app.registration** — Use dot representation if you want to use complete name |
| Class | Try to name the class by suffixing according to the package in which it is created or a meaning-full name which is easy to remember and can be find using a regex. | • PendingNotice**Resource**.java<br>• PendingNotice**Service**.java<br>• PendingNotice**ServiceImpl**.java<br>• PendingNotice**Repository**.java<br>• PendingNotice**RepositoryImpl**.java<br>• PendingNotice**Mapper**.java<br>• PendingNotice**Request**.java<br>• PendingNotice**Response**.java |
| Entities | Entities class name should be suffixed with *Entity*. | • DcCase**Entity**.java<br>• ArIndividual**Entity**.java |
| DTO | DTO class name should be suffixed with *Dto*. | • DcCase**Dto**.java<br>• ArIndividual**Dto**.java |
| Mappers | Mapper interface name should be suffixed with *Mapper*. | • DcCase**Mapper**.java<br>• ArIndividual**Mapper**.java |
| Utility classes | Utility class name should suffix Utility at the end of the file name. | • Date**Utility**.java<br>• String**Utility**.java<br>• Number**Utility**.java |

## 4.2. Rest APIs

Creating web service end points is an important task, as it help developers to understand what it consumes and produces. The URI which we create needs to be logical and self-explanatory, for this will follow rest convention to create APIs.

I have taken extract from the following link to follow resource naming convention.

### 4.2.1. Versioning

Use URI versioning when creating URI, doing this we can give different implementation for the same business scenario. Also, this help in maintaining backward compatible URIs and slowly deprecating the currently used URIs.

**Versioning Scenario**

Suppose you have a URI which produces individual data that return only first_name and last_name in response.

- http://api.example.com/v1/data-collection/individual/{id}

Now, there is a requirement where we need to change the response structure and still we need to have the above URI. In this case you can create a new version with updated URI as below.

- http://api.example.com/v2/data-collection/individual/{id}

### 4.2.2. Naming Convention

**Use nouns to represent resources**

RESTful URI should refer to a resource that is a thing (noun) instead of referring to an action (verb) because nouns have properties which verbs do not have – similar to resources have attributes. Some examples of a resource and their resource URIs can be designed as below

- Users of the system
- User Notices
- Case Task

```
http://api.example.com/v1/correspondence/notices/{id}
http://api.example.com/v1/admin/tasks
http://api.example.com/v1/data-collection/individual/{id}
http://api.example.com/v1/data-collection/case/individual/{id}
http://api.example.com/v1/data-collection/case //Send case number in request body
http://api.example.com/v1/user-management/users/{id}
```

- Never create an endpoint where you are sending PII data in URL as GET params, and as a variable in an endpoint.
- Create a POST request and send data in body when we are dealing with PII.
- Developers can use unique_trans_id to fetch resource data.

**Never use CRUD function names in URIs**

URIs should not be used to indicate that what CRUD operation is being perform. URIs should be used to uniquely identify resources and not any action upon them. HTTP request methods should be used to indicate which CRUD function is performed.

```
HTTP GET http://api.example.com/v1/correspondence/notices  ①
HTTP POST http://api.example.com/v1/correspondence/notices  ②

HTTP GET http://api.example.com/v1/data-collection/individual/income/{id} ③
HTTP PUT http://api.example.com/v1/data-collection/individual/income/{id} ④
HTTP DELETE http://api.example.com/v1/data-collection/individual/income/{id} ⑤
```

① Get all notices.

② Create new notice.

③ Get individual’s income for given Id.

④ Update individual’s income for given Id.

⑤ Delete individual’s income for given Id.

### 4.2.3. Endpoints

This section will help you in understanding how to name an endpoint. We recommend following points to keep in mind while creating an endpoint.

1. Name the rest controller path with what type of operation it’s going to perform. Please refer below examples
   If the controller is going to perform operations on an individual then name the root path as **"/individual"**
   If the controller is going to perform operations on a notice then name the root path as **"/notice"**
2. Name individual endpoint according to what operation it does and do not use method name in the URI endpoint. Refer Naming convention on how each endpoint should be named.
3. Never pass PII data in request params or path variables.
4. Try to use combination of HTTP methods and method overloading pattern of java.

*Incorrect way of naming the endpoints*

```
@RestController
@RequestMapping("/appIndvFetch") ①
public class AppIndvFetchController {

    private final ApplicationIndvFetchService applicationIndvFetchService;

    @PostMapping("/fetchAppIndv") # <.> ② ③
    public ResponseEntity<ApiResponse> fetchAppIndv(@RequestBody AppIndvListRequest
appIndvListRequest)  { ④
        ...
    }
}
```

① It does not help to understand what this controller is about and what functions it’s going to perform.

② Exposes method name as URI endpoint.

③ Incorrect HTTP method used.

④ No versioning done.

⑤ No validation annotation added.

*Correct way of naming the endpoints*

```
@RestController
@RequestMapping("/search") ①
public class AppIndvFetchController {

    private final ApplicationIndvFetchService applicationIndvFetchService;


    @GetMapping("/") ②
    public ResponseEntity<ApiResponse> search(@RequestParam String appNum,
@RequestParam boolean activeSw)  { ③
        ...
    }

    @PostMapping("/") ④
    public ResponseEntity<ApiResponse> search(@Valid @RequestBody SearchRequest
searchRequest)  { ⑤
        ...
    }

    @GetMapping("/individual/{appNum}") ⑥
    public ResponseEntity<ApiResponse> search(@NotNull @PathVariable String appNum)  {
        ...
    }
}
```

① By reading the root path we understand that this controller has functionality related to search.

② This endpoint says that you can search application in respective project with passing two request params.

③ If endpoint have more than 2 or 3 request param we recommend creating a POST request and passing data in request body. Here we have repeated same mapping path as in point 1, but the difference is in HTTP methods and method signature.

④ This endpoint says that we can search all individual belonging to application number.

### Anti-patterns

1. Anti-Pattern Examples

| Anti-pattern URI | Reason | Resolution |
|---|---|---|

| | | |
|---|---|---|
| *POST* http://api.example.com/app_reg/appIndvFetch/fetchAppIndv | • Endpoint name is too verbose.<br>• Exposes method name using in the file.<br>• No versioning. | *GET* http://api.example.com/app_reg/api/v1/indiv/{app_num} |
| *POST* http://api.example.com/app_reg/appsearch/findApps | • Same synonyms use (e.g. search, find)<br>• We already have context as app-reg so we know where we are searching. No need to mention *"appsearch"* again.<br>• No versioning. | *GET* http://api.example.com/app_reg/api/v1/search/{app_num} |
| *POST* http://api.example.com/app_reg/api/application/findAuthRepDetailsForAppNum/{appNum}/{caseNum} | • Exposing case number which is a PII.<br>• Inconsistent with above URI endpoint created in same application. The above URIs do not start start with **/api/**<br>• If a developer use same name with 2 path variable if will confuse the Spring framework and results in error. | • *GET* http://api.example.com/app_reg/api/v1/authRep/{app_num}<br>• *POST* http://api.example.com/app_reg/api/v1/authRep *//Pass case number in request body* |
| *POST* http://api.example.com/app_reg/createAuthRep | • Exposing function name in URI endpoint. | POST http://api.example.com/app_reg/api/v1/authRep *//Pass auth rep data in request body* |
| *GET* http://api.example.com/app_reg/deleteAuthRepDataForAppNum/{app_num} | • Exposing function name in URI endpoint.<br>• Incorrect HTTP method used. Here we are deleting a record and using GET method instead of DELETE method | *DELETE* http://api.example.com/app_reg/api/v1/authRep/{unique_id_of_authRep_table} |

| | | |
|---|---|---|
| *POST* http://api.example.com/app_reg/api/application/associateCase/{appNum}/{caseNum} | • Exposing case number which is a PII.<br>• The URL endpoint name is too verbose and expose method name. | *POST* http://api.example.com/app_reg/api/v1/application/associate *//Send app_num and case_num in request body* |
| *POST* http://api.example.com/correspondence/api/correspondence/generateManual/caseSearch | • Context root is repeated.<br>• Developer know they are searching notices for a case, no need to say what this endpoint is going to search. You can always pass generate_manual_sw in request body to get a specific type of notice.<br>• There should be always a global search in respective applications and URI should not be specific as in this example. | *POST* http://api.example.com/correspondence/api/v1/search *//Send case_num in request body* |
| *POST* http://api.example.com/correspondence/api/correspondence/massMailing/createOrUpdateRequest | • This URI violates single use principle, it does two different this here create and update<br>• Context root is repeated.<br>• Updates need to be a PUT request | *POST* http://api.example.com/correspondence/api/v1/mailing<br>*PUT* http://api.example.com/correspondence/api/v1/mailing/{id} |
| *POST* http://api.example.com/correspondence/api/correspondence/massMailing/delete | • The URI business operation contradicts with HTTP method. | *DELETE* http://api.example.com/correspondence/api/v1/massMailing/{id} |

## 5. Keyboard Shortcuts

Use below shortcuts during development.

| **IntelliJ** | **Eclipse** |
|---|---|
| `Ctrl` + `Space` for suggestion. | `Ctrl` + `Space` for suggestion |
| `Ctrl` + `Alt` + `L` to format code. | `Ctrl` + `Shift` + `F` to format code. |
| Double click `Shift` to search any file in work space. | `Ctrl` + `Shift` + `R` search for resource. |

| IntelliJ | Eclipse |
|---|---|
| `Ctrl` + `F12` to view methods in file. | `Ctrl` + `O` to view methods in file. |
| `Ctrl` + `O` to override/implement methods. | |
| `Ctrl` + `Alt` + `Shift` + `L` to reformat file. | `Ctrl` + `SHFIT` + `O` Organize imports. |

# 6. Framework Modules