```
/*
 * adonis-responsive-attachment
 *
 * (c) Ndianabasi Udonkang <ndianabasi@furnish.ng>
 *
 * For the full copyright and license information, please view the LICENSE
 * file that was distributed with this source code.
 */

import 'reflect-metadata'

import test from 'japa'
import { join } from 'path'
import supertest from 'supertest'
import { createServer } from 'http'
import { ApplicationContract } from '@ioc:Adonis/Core/Application'
import { BodyParserMiddleware } from '@adonisjs/bodyparser/build/src/BodyParser'
import { getDimensions } from '../src/Helpers/ImageManipulationHelper'

import { ResponsiveAttachment } from '../src/Attachment'
import { setup, cleanup, setupApplication } from '../test-helpers'
import { readFile } from 'fs/promises'

let app: ApplicationContract

const samplePersistedImageData = {
  isPersisted: true,
  isLocal: false,
  name: 'original_ckw5lpv7v0002egvobe1b0oav.jpg',
  size: 291.69,
  hash: 'ckw5lpv7v0002egvobe1b0oav',
  width: 1500,
  format: 'jpeg',
  height: 1000,
  extname: 'jpg',
  mimeType: 'image/jpeg',
  url: '/uploads/original_ckw5lpv7v0002egvobe1b0oav.jpg?
signature=eyJtZXNzYWdlIjoiL3VwbG9hZHMvb3JpZ2luYWxfY2t3NWxwdjd2MDAwMmVndm
9iZTFiMG9hdi5qcGcifQ.ieXMlaRb8izlREvJ0E9iMY0I3iedalmv-pvOUIrfEZc',
  breakpoints: {
    thumbnail: {
      name: 'thumbnail_ckw5lpv7v0002egvobe1b0oav.jpg',
      hash: 'ckw5lpv7v0002egvobe1b0oav',
      extname: 'jpg',
      mimeType: 'image/jpeg',
      width: 234,
      height: 156,
      size: 7.96,
      url: '/uploads/thumbnail_ckw5lpv7v0002egvobe1b0oav.jpg?
signature=eyJtZXNzYWdlIjoiL3VwbG9hZHMvdGh1bWJuYWlsX2NrdzVscHY3djAwMDJlZ3Zv
YmUxYjBvYXYuanBnIn0.RGGimHh6NuyPrB2ZgmudE7rH4RRCT3NL7kex9EmSyIo',
    },
```

```
    large: {
      name: 'large_ckw5lpv7v0002egvobe1b0oav.jpg',
      hash: 'ckw5lpv7v0002egvobe1b0oav',
      extname: 'jpg',
      mimeType: 'image/jpeg',
      width: 1000,
      height: 667,
      size: 129.15,
      url: '/uploads/large_ckw5lpv7v0002egvobe1b0oav.jpg?
signature=eyJtZXNzYWdlIjoiL3VwbG9hZHMvbGFyZ2VfY2t3NWxwdjd2MDAwMmVndm9iZT
FiMG9hdi5qcGcifQ.eNC8DaqYCYd4khKhqS7DKI66SsLpD-vyVIaP8rzMmAA',
    },
    medium: {
      name: 'medium_ckw5lpv7v0002egvobe1b0oav.jpg',
      hash: 'ckw5lpv7v0002egvobe1b0oav',
      extname: 'jpg',
      mimeType: 'image/jpeg',
      width: 750,
      height: 500,
      size: 71.65,
      url: '/uploads/medium_ckw5lpv7v0002egvobe1b0oav.jpg?
signature=eyJtZXNzYWdlIjoiL3VwbG9hZHMvbWVkaXVtX2NrdzVscHY3djAwMDJlZ3ZvYmU
xYjBvYXYuanBnIn0.2ADmssxFC0vxmq4gJEgjb9Fxo1qcQ6tMVeKBqZ1ENkM',
    },
    small: {
      name: 'small_ckw5lpv7v0002egvobe1b0oav.jpg',
      hash: 'ckw5lpv7v0002egvobe1b0oav',
      extname: 'jpg',
      mimeType: 'image/jpeg',
      width: 500,
      height: 333,
      size: 32.21,
      url: '/uploads/small_ckw5lpv7v0002egvobe1b0oav.jpg?
signature=eyJtZXNzYWdlIjoiL3VwbG9hZHMvc21hbGxfY2t3NWxwdjd2MDAwMmVndm9iZTF
iMG9hdi5qcGcifQ.I8fwMRwY5azvlS_8B0K40BWKQNLuS-HqCB_3RXryOok',
    },
  },
}

test.group('ResponsiveAttachment | fromDbResponse', (group) => {
  group.before(async () => {
    app = await setupApplication()
    await setup(app)

    app.container.resolveBinding('Adonis/Core/Route').commit()
    ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
  })

  group.after(async () => {
    await cleanup(app)
  })
```

```
  test('create image attachment instance from db response', (assert) => {
    const responsiveAttachment = ResponsiveAttachment.fromDbResponse(
      JSON.stringify(samplePersistedImageData)
    )

    assert.isTrue(responsiveAttachment?.isPersisted)
    assert.isFalse(responsiveAttachment?.isLocal)
  })

  test('save method should result in noop when image attachment is created from db response', async
(assert) => {
    const responsiveAttachment = ResponsiveAttachment.fromDbResponse(
      JSON.stringify(samplePersistedImageData)
    )

    await responsiveAttachment?.save()
    assert.equal(responsiveAttachment?.name, 'original_ckw5lpv7v0002egvobe1b0oav.jpg')
    assert.equal(
      responsiveAttachment?.breakpoints?.thumbnail.name,
      'thumbnail_ckw5lpv7v0002egvobe1b0oav.jpg'
    )
    assert.equal(
      responsiveAttachment?.breakpoints?.small.name,
      'small_ckw5lpv7v0002egvobe1b0oav.jpg'
    )
    assert.equal(
      responsiveAttachment?.breakpoints?.medium.name,
      'medium_ckw5lpv7v0002egvobe1b0oav.jpg'
    )
    assert.equal(
      responsiveAttachment?.breakpoints?.large.name,
      'large_ckw5lpv7v0002egvobe1b0oav.jpg'
    )
  })

  test('delete persisted images', async (assert) => {
    const responsiveAttachment = ResponsiveAttachment.fromDbResponse(
      JSON.stringify(samplePersistedImageData)
    )

    await responsiveAttachment?.delete()
    assert.isTrue(responsiveAttachment?.isDeleted)
  })

  test('compute image urls', async (assert) => {
    const responsiveAttachment = ResponsiveAttachment.fromDbResponse(
      JSON.stringify(samplePersistedImageData)
    )

    responsiveAttachment?.setOptions({ preComputeUrls: true })

    await responsiveAttachment?.computeUrls()
```

```
  assert.match(
    responsiveAttachment?.url!,
    /\/uploads\/original_ckw5lpv7v0002egvobe1b0oav\.jpg\?signature=/
  )
  assert.match(
    responsiveAttachment?.breakpoints?.thumbnail.url!,
    /\/uploads\/thumbnail_ckw5lpv7v0002egvobe1b0oav\.jpg\?signature=/
  )
  assert.match(
    responsiveAttachment?.breakpoints?.small.url!,
    /\/uploads\/small_ckw5lpv7v0002egvobe1b0oav\.jpg\?signature=/
  )
  assert.match(
    responsiveAttachment?.breakpoints?.large.url!,
    /\/uploads\/large_ckw5lpv7v0002egvobe1b0oav\.jpg\?signature=/
  )
  assert.match(
    responsiveAttachment?.breakpoints?.medium.url!,
    /\/uploads\/medium_ckw5lpv7v0002egvobe1b0oav\.jpg\?signature=/
  )
})

test('compute image urls from a custom method', async (assert) => {
  const responsiveAttachment = ResponsiveAttachment.fromDbResponse(
    JSON.stringify(samplePersistedImageData)
  )

  responsiveAttachment?.setOptions({
    preComputeUrls: async (_, image: ResponsiveAttachment) => {
      return {
        url: `/custom_folder${image.url!}`,
        breakpoints: {
          thumbnail: {
            url: `/custom_folder${image.breakpoints?.thumbnail.url!}`,
          },
          small: { url: `/custom_folder${image.breakpoints?.small.url!}` },
          medium: { url: `/custom_folder${image.breakpoints?.medium.url!}` },
          large: { url: `/custom_folder${image.breakpoints?.large.url!}` },
        },
      }
    },
  })

  await responsiveAttachment?.computeUrls()

  assert.match(
    responsiveAttachment?.url!,
    /\/custom_folder\/uploads\/original_ckw5lpv7v0002egvobe1b0oav\.jpg\?signature=/
  )
  assert.match(
    responsiveAttachment?.urls?.breakpoints?.small.url!,
```

```
      /\/custom_folder\/uploads\/small_ckw5lpv7v0002egvobe1b0oav\.jpg\?signature=/
    )
    assert.match(
      responsiveAttachment?.urls?.breakpoints?.large.url!,
      /\/custom_folder\/uploads\/large_ckw5lpv7v0002egvobe1b0oav\.jpg\?signature=/
    )
    assert.match(
      responsiveAttachment?.urls?.breakpoints?.medium.url!,
      /\/custom_folder\/uploads\/medium_ckw5lpv7v0002egvobe1b0oav\.jpg\?signature=/
    )
  })
})

test.group('ResponsiveAttachment | fromFile', (group) => {
  group.before(async () => {
    app = await setupApplication()
    await setup(app)

    app.container.resolveBinding('Adonis/Core/Route').commit()
    ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
  })

  group.after(async () => {
    await cleanup(app)
  })

  test('create attachment from the user-uploaded image', async (assert) => {
    const Drive = app.container.resolveBinding('Adonis/Core/Drive')

    const server = createServer((req, res) => {
      const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)
      app.container.make(BodyParserMiddleware).handle(ctx, async () => {
        const file = ctx.request.file('avatar')!
        const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
        await responsiveAttachment?.save()

        assert.isTrue(responsiveAttachment?.isPersisted)
        assert.isTrue(responsiveAttachment?.isLocal)

        assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
        assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.thumbnail.name!))
        assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.small.name!))
        assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.medium.name!))
        assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.large.name!))

        ctx.response.send(responsiveAttachment)
        ctx.response.finish()
      })
    })

    const { body } = await supertest(server)
      .post('/')
```

```
    .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg'))

  assert.isTrue(await Drive.exists(body.name))
  assert.isTrue(await Drive.exists(body.breakpoints?.thumbnail.name))
  assert.isTrue(await Drive.exists(body.breakpoints?.small.name))
  assert.isTrue(await Drive.exists(body.breakpoints?.medium.name))
  assert.isTrue(await Drive.exists(body.breakpoints?.large.name))
})

test('change the format of the user-uploaded image', async (assert) => {
  const server = createServer((req, res) => {
    const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)
    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const file = ctx.request.file('avatar')!
      const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
      responsiveAttachment?.setOptions({ preComputeUrls: true, forceFormat: 'webp' })
      await responsiveAttachment?.save()

      assert.isTrue(responsiveAttachment?.isPersisted)
      assert.isTrue(responsiveAttachment?.isLocal)

      assert.include(responsiveAttachment?.name!, 'webp')
      assert.include(responsiveAttachment?.breakpoints?.thumbnail.name!, 'webp')
      assert.include(responsiveAttachment?.breakpoints?.small.name!, 'webp')
      assert.include(responsiveAttachment?.breakpoints?.medium.name!, 'webp')
      assert.include(responsiveAttachment?.breakpoints?.large.name!, 'webp')

      assert.include(responsiveAttachment?.url!, 'webp')
      assert.include(responsiveAttachment?.urls?.breakpoints?.large.url!, 'webp')
      assert.include(responsiveAttachment?.urls?.breakpoints?.medium.url!, 'webp')
      assert.include(responsiveAttachment?.urls?.breakpoints?.small.url!, 'webp')
      assert.include(responsiveAttachment?.urls?.breakpoints?.thumbnail.url!, 'webp')

      ctx.response.send(responsiveAttachment)
      ctx.response.finish()
    })
  })

  await supertest(server)
    .post('/')
    .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg'))
})

test('pre-compute urls for newly-created images', async (assert) => {
  const Drive = app.container.resolveBinding('Adonis/Core/Drive')

  const server = createServer((req, res) => {
    const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)

    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const file = ctx.request.file('avatar')!
      const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
```

```
    responsiveAttachment?.setOptions({ preComputeUrls: true })
    await responsiveAttachment?.save()

    assert.isTrue(responsiveAttachment?.isPersisted)
    assert.isTrue(responsiveAttachment?.isLocal)

    assert.isNotEmpty(responsiveAttachment?.urls)

    assert.isDefined(responsiveAttachment?.url)
    assert.isDefined(responsiveAttachment?.urls?.breakpoints?.large.url)
    assert.isDefined(responsiveAttachment?.urls?.breakpoints?.medium.url)
    assert.isDefined(responsiveAttachment?.urls?.breakpoints?.small.url)
    assert.isDefined(responsiveAttachment?.urls?.breakpoints?.thumbnail.url)

    assert.isNotNull(responsiveAttachment?.url)
    assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.large.url)
    assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.medium.url)
    assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.small.url)
    assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.thumbnail.url)

    assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
    assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.large.name!))
    assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.medium.name!))
    assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.small.name!))
    assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.thumbnail.name!))

    assert.isTrue(
      responsiveAttachment?.breakpoints!.thumbnail.size! <
        responsiveAttachment?.breakpoints!.small.size!
    )
    assert.isTrue(
      responsiveAttachment?.breakpoints!.small.size! <
        responsiveAttachment?.breakpoints!.medium.size!
    )
    assert.isTrue(
      responsiveAttachment?.breakpoints!.medium.size! <
        responsiveAttachment?.breakpoints!.large.size!
    )
    assert.isTrue(responsiveAttachment?.breakpoints!.large.size! < responsiveAttachment?.size!)

    ctx.response.send(responsiveAttachment)
    ctx.response.finish()
   })
  })

  await supertest(server)
    .post('/')
    .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg'))
})

test('delete local images', async (assert) => {
  const Drive = app.container.resolveBinding('Adonis/Core/Drive')
```

```
  const server = createServer((req, res) => {
    const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)

    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const file = ctx.request.file('avatar')!
      const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
      responsiveAttachment?.setOptions({ preComputeUrls: true })
      await responsiveAttachment?.save()
      await responsiveAttachment?.delete()

      assert.isFalse(responsiveAttachment?.isPersisted)
      assert.isTrue(responsiveAttachment?.isLocal)
      assert.isTrue(responsiveAttachment?.isDeleted)

      ctx.response.send(responsiveAttachment)
      ctx.response.finish()
    })
  })

  const { body } = await supertest(server)
    .post('/')
    .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg'))

  assert.isDefined(body.url)
  assert.isDefined(body.breakpoints.large.url)
  assert.isDefined(body.breakpoints.medium.url)
  assert.isDefined(body.breakpoints.small.url)
  assert.isDefined(body.breakpoints.thumbnail.url)

  assert.isNotNull(body.url)
  assert.isNotNull(body.breakpoints.large.url)
  assert.isNotNull(body.breakpoints.medium.url)
  assert.isNotNull(body.breakpoints.small.url)
  assert.isNotNull(body.breakpoints.thumbnail.url)

  assert.isFalse(await Drive.exists(body.name))
  assert.isFalse(await Drive.exists(body.breakpoints.large.name))
  assert.isFalse(await Drive.exists(body.breakpoints.medium.name))
  assert.isFalse(await Drive.exists(body.breakpoints.small.name))
  assert.isFalse(await Drive.exists(body.breakpoints.thumbnail.name))
})

test('do not create any image when image is not attached', async (assert) => {
  const server = createServer((req, res) => {
    const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)
    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const file = ctx.request.file('avatar')!
      const responsiveAttachment = file ? await ResponsiveAttachment.fromFile(file) : null
      await responsiveAttachment?.save()

      assert.isNull(responsiveAttachment)
```

```
        ctx.response.send(responsiveAttachment)
        ctx.response.finish()
      })
    })

    const { body } = await supertest(server).post('/')

    assert.isEmpty(body)
  })
})

test.group('ImageManipulationHelper', (group) => {
  group.before(async () => {
    app = await setupApplication()
    await setup(app)

    app.container.resolveBinding('Adonis/Core/Route').commit()
    ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
  })

  group.after(async () => {
    await cleanup(app)
  })

  test('getDimensions', async (assert) => {
    const Drive = app.container.resolveBinding('Adonis/Core/Drive')

    const server = createServer((req, res) => {
      const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)
      app.container.make(BodyParserMiddleware).handle(ctx, async () => {
        const file = ctx.request.file('avatar')!
        await file.moveToDisk('image_upload_tmp')
        const buffer = await Drive.get(file.filePath!)
        const { width, height } = await getDimensions(buffer)

        assert.equal(width, 1500)
        assert.equal(height, 1000)

        ctx.response.send({ width, height })
        ctx.response.finish()
      })
    })

    const {
      body: { width, height },
    } = await supertest(server)
      .post('/')
      .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg'))

    assert.equal(width, 1500)
    assert.equal(height, 1000)
```

```
  })
})

test.group('Images below the thumbnail resize options', (group) => {
  group.before(async () => {
    app = await setupApplication()
    await setup(app)

    app.container.resolveBinding('Adonis/Core/Route').commit()
    ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
  })

  group.after(async () => {
    await cleanup(app)
  })

  test('does not create thumbnail and responsive images for files below the
THUMBNAIL_RESIZE_OPTIONS', async (assert) => {
    const Drive = app.container.resolveBinding('Adonis/Core/Drive')

    const server = createServer((req, res) => {
      const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)
      app.container.make(BodyParserMiddleware).handle(ctx, async () => {
        const file = ctx.request.file('avatar')!
        const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
        await responsiveAttachment?.save()

        assert.isTrue(responsiveAttachment?.isPersisted)
        assert.isTrue(responsiveAttachment?.isLocal)

        assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
        assert.isUndefined(responsiveAttachment?.breakpoints)

        ctx.response.send(responsiveAttachment)
        ctx.response.finish()
      })
    })

    await supertest(server)
      .post('/')
      .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-150x100.jpg'))
  })

  test('pre-compute urls for newly created image below the THUMBNAIL_RESIZE_OPTIONS',
async (assert) => {
    const Drive = app.container.resolveBinding('Adonis/Core/Drive')

    const server = createServer((req, res) => {
      const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)

      app.container.make(BodyParserMiddleware).handle(ctx, async () => {
        const file = ctx.request.file('avatar')!
```

```
        const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
        responsiveAttachment?.setOptions({ preComputeUrls: true })
        await responsiveAttachment?.save()

        assert.isTrue(responsiveAttachment?.isPersisted)
        assert.isTrue(responsiveAttachment?.isLocal)

        assert.isNotEmpty(responsiveAttachment?.urls)

        assert.isDefined(responsiveAttachment?.url)
        assert.isNotNull(responsiveAttachment?.url)

        assert.isUndefined(responsiveAttachment?.urls?.breakpoints)

        assert.isTrue(await Drive.exists(responsiveAttachment?.name!))

        ctx.response.send(responsiveAttachment)
        ctx.response.finish()
      })
    })

    await supertest(server)
      .post('/')
      .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-150x100.jpg'))
  })
})

test.group('Images below the large breakeven point', (group) => {
  group.before(async () => {
    app = await setupApplication()
    await setup(app)

    app.container.resolveBinding('Adonis/Core/Route').commit()
    ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
  })

  group.after(async () => {
    await cleanup(app)
  })

  test('create attachment from the user uploaded image', async (assert) => {
    const Drive = app.container.resolveBinding('Adonis/Core/Drive')

    const server = createServer((req, res) => {
      const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)
      app.container.make(BodyParserMiddleware).handle(ctx, async () => {
        const file = ctx.request.file('avatar')!
        const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
        await responsiveAttachment?.save()

        assert.isTrue(responsiveAttachment?.isPersisted)
        assert.isTrue(responsiveAttachment?.isLocal)
```

```
      assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.thumbnail.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.small.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.medium.name!))

      assert.isUndefined(responsiveAttachment?.breakpoints?.large)

      ctx.response.send(responsiveAttachment)
      ctx.response.finish()
    })
  })

  await supertest(server)
    .post('/')
    .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-825x550.jpg'))
})

test('pre-compute urls for newly created images', async (assert) => {
  const Drive = app.container.resolveBinding('Adonis/Core/Drive')

  const server = createServer((req, res) => {
    const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)

    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const file = ctx.request.file('avatar')!
      const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
      responsiveAttachment?.setOptions({ preComputeUrls: true })
      await responsiveAttachment?.save()

      assert.isTrue(responsiveAttachment?.isPersisted)
      assert.isTrue(responsiveAttachment?.isLocal)

      assert.isNotEmpty(responsiveAttachment?.urls)

      assert.isDefined(responsiveAttachment?.url)
      assert.isDefined(responsiveAttachment?.urls?.breakpoints?.medium.url)
      assert.isDefined(responsiveAttachment?.urls?.breakpoints?.small.url)
      assert.isDefined(responsiveAttachment?.urls?.breakpoints?.thumbnail.url)

      assert.isUndefined(responsiveAttachment?.urls?.breakpoints?.large)

      assert.isNotNull(responsiveAttachment?.url)
      assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.medium.url)
      assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.small.url)
      assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.thumbnail.url)

      assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.medium.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.small.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.thumbnail.name!))
```

```
      assert.isUndefined(responsiveAttachment?.breakpoints?.large)

      assert.isTrue(
        responsiveAttachment?.breakpoints!.thumbnail.size! <
          responsiveAttachment?.breakpoints!.small.size!
      )
      assert.isTrue(
        responsiveAttachment?.breakpoints!.small.size! <
          responsiveAttachment?.breakpoints!.medium.size!
      )
      assert.isTrue(responsiveAttachment?.breakpoints!.medium.size! <
responsiveAttachment?.size!)

      ctx.response.send(responsiveAttachment)
      ctx.response.finish()
    })
  })

  await supertest(server)
    .post('/')
    .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-825x550.jpg'))
 })
})

test.group(
 'Do not generate responsive images when `options.responsiveDimensions` is false',
 (group) => {
  group.before(async () => {
    app = await setupApplication()
    await setup(app)

    app.container.resolveBinding('Adonis/Core/Route').commit()
    ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
  })

  group.after(async () => {
    await cleanup(app)
  })

  test('create attachment from the user uploaded image', async (assert) => {
    const Drive = app.container.resolveBinding('Adonis/Core/Drive')

    const server = createServer((req, res) => {
      const ctx = app.container
        .resolveBinding('Adonis/Core/HttpContext')
        .create('/', {}, req, res)
      app.container.make(BodyParserMiddleware).handle(ctx, async () => {
        const file = ctx.request.file('avatar')!
        const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
        responsiveAttachment?.setOptions({ responsiveDimensions: false })
        await responsiveAttachment?.save()
```

```
        assert.isTrue(responsiveAttachment?.isPersisted)
        assert.isTrue(responsiveAttachment?.isLocal)

        assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
        assert.isUndefined(responsiveAttachment?.breakpoints)

        ctx.response.send(responsiveAttachment)
        ctx.response.finish()
      })
    })

    await supertest(server)
      .post('/')
      .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-825x550.jpg'))
  })

  test('pre-compute urls for newly created images', async (assert) => {
    const Drive = app.container.resolveBinding('Adonis/Core/Drive')

    const server = createServer((req, res) => {
      const ctx = app.container
        .resolveBinding('Adonis/Core/HttpContext')
        .create('/', {}, req, res)

      app.container.make(BodyParserMiddleware).handle(ctx, async () => {
        const file = ctx.request.file('avatar')!
        const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
        responsiveAttachment?.setOptions({ preComputeUrls: true, responsiveDimensions: false })
        await responsiveAttachment?.save()

        assert.isTrue(responsiveAttachment?.isPersisted)
        assert.isTrue(responsiveAttachment?.isLocal)

        assert.isNotEmpty(responsiveAttachment?.urls)

        assert.isDefined(responsiveAttachment?.url)
        assert.isNotNull(responsiveAttachment?.url)
        assert.isUndefined(responsiveAttachment?.urls?.breakpoints)

        assert.isTrue(await Drive.exists(responsiveAttachment?.name!))

        ctx.response.send(responsiveAttachment)
        ctx.response.finish()
      })
    })

    await supertest(server)
      .post('/')
      .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-825x550.jpg'))
  })
}
)
```

```
test.group('Do not generate thumbnail images when `options.disableThumbnail` is true', (group) =>
{
  group.before(async () => {
    app = await setupApplication()
    await setup(app)

    app.container.resolveBinding('Adonis/Core/Route').commit()
    ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
  })

  group.after(async () => {
    await cleanup(app)
  })

  test('create attachment from the user uploaded image', async (assert) => {
    const Drive = app.container.resolveBinding('Adonis/Core/Drive')

    const server = createServer((req, res) => {
      const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)
      app.container.make(BodyParserMiddleware).handle(ctx, async () => {
        const file = ctx.request.file('avatar')!
        const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
        responsiveAttachment?.setOptions({ disableThumbnail: true })
        await responsiveAttachment?.save()

        assert.isTrue(responsiveAttachment?.isPersisted)
        assert.isTrue(responsiveAttachment?.isLocal)

        assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
        assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.small.name!))
        assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.medium.name!))
        assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.large.name!))

        assert.isUndefined(responsiveAttachment?.breakpoints?.thumbnail)

        ctx.response.send(responsiveAttachment)
        ctx.response.finish()
      })
    })

    await supertest(server)
      .post('/')
      .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg'))
  })

  test('pre-compute urls for newly created images', async (assert) => {
    const server = createServer((req, res) => {
      const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)

      app.container.make(BodyParserMiddleware).handle(ctx, async () => {
        const file = ctx.request.file('avatar')!
```

```
        const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
        responsiveAttachment?.setOptions({ preComputeUrls: true, disableThumbnail: true })
        await responsiveAttachment?.save()

        assert.isTrue(responsiveAttachment?.isPersisted)
        assert.isTrue(responsiveAttachment?.isLocal)

        assert.isNotEmpty(responsiveAttachment?.urls)

        assert.isDefined(responsiveAttachment?.url)
        assert.isNotNull(responsiveAttachment?.url)
        assert.isDefined(responsiveAttachment?.url!)
        assert.isDefined(responsiveAttachment?.url!)
        assert.isDefined(responsiveAttachment?.breakpoints?.small.url!)
        assert.isNotNull(responsiveAttachment?.breakpoints?.small.url!)
        assert.isDefined(responsiveAttachment?.breakpoints?.medium.url!)
        assert.isNotNull(responsiveAttachment?.breakpoints?.medium.url!)
        assert.isDefined(responsiveAttachment?.breakpoints?.large.url!)
        assert.isNotNull(responsiveAttachment?.breakpoints?.large.url!)

        assert.isUndefined(responsiveAttachment?.breakpoints?.thumbnail)

        ctx.response.send(responsiveAttachment)
        ctx.response.finish()
      })
    })

    await supertest(server)
      .post('/')
      .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg'))
  })
})

test.group('Do not generate responsive images when some default breakpoints are `off`', (group) =>
{
  group.before(async () => {
    app = await setupApplication()
    await setup(app)

    app.container.resolveBinding('Adonis/Core/Route').commit()
    ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
  })

  group.after(async () => {
    await cleanup(app)
  })

  test('create attachment from the user uploaded image', async (assert) => {
    const Drive = app.container.resolveBinding('Adonis/Core/Drive')

    const server = createServer((req, res) => {
      const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)
```

```
    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const file = ctx.request.file('avatar')!
      const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
      responsiveAttachment?.setOptions({
        breakpoints: { medium: 'off', small: 'off' },
      })
      await responsiveAttachment?.save()

      assert.isTrue(responsiveAttachment?.isPersisted)
      assert.isTrue(responsiveAttachment?.isLocal)

      assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.large.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.thumbnail.name!))

      assert.isUndefined(responsiveAttachment?.breakpoints?.medium)
      assert.isUndefined(responsiveAttachment?.breakpoints?.small)

      assert.isTrue(
        responsiveAttachment?.breakpoints!.thumbnail.size! <
          responsiveAttachment?.breakpoints!.large.size!
      )
      assert.isTrue(responsiveAttachment?.breakpoints!.large.size! < responsiveAttachment?.size!)

      ctx.response.send(responsiveAttachment)
      ctx.response.finish()
    })
  })

  await supertest(server)
    .post('/')
    .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg'))
})

test('pre-compute urls for newly created images', async (assert) => {
  const server = createServer((req, res) => {
    const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)

    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const file = ctx.request.file('avatar')!
      const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
      responsiveAttachment?.setOptions({
        preComputeUrls: true,
        breakpoints: { medium: 'off', small: 'off' },
      })
      await responsiveAttachment?.save()

      assert.isTrue(responsiveAttachment?.isPersisted)
      assert.isTrue(responsiveAttachment?.isLocal)

      assert.isNotEmpty(responsiveAttachment?.urls)
```

```
        assert.isDefined(responsiveAttachment?.url)
        assert.isNotNull(responsiveAttachment?.url)
        assert.isDefined(responsiveAttachment?.breakpoints?.large.url!)
        assert.isDefined(responsiveAttachment?.breakpoints?.thumbnail.url!)
        assert.isNotNull(responsiveAttachment?.breakpoints?.large.url!)
        assert.isNotNull(responsiveAttachment?.breakpoints?.thumbnail.url!)

        ctx.response.send(responsiveAttachment)
        ctx.response.finish()
      })
    })

    await supertest(server)
      .post('/')
      .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg'))
  })
})

test.group('Manual generation of URLs', (group) => {
  group.before(async () => {
    app = await setupApplication()
    await setup(app)

    app.container.resolveBinding('Adonis/Core/Route').commit()
    ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
  })

  group.after(async () => {
    await cleanup(app)
  })

  test('generate URLs for the images', async (assert) => {
    const server = createServer((req, res) => {
      const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)
      app.container.make(BodyParserMiddleware).handle(ctx, async () => {
        const file = ctx.request.file('avatar')!
        const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
        responsiveAttachment?.setOptions(undefined)
        await responsiveAttachment?.save()
        const urls = await responsiveAttachment?.getUrls()

        assert.isTrue(responsiveAttachment?.isPersisted)
        assert.isTrue(responsiveAttachment?.isLocal)

        assert.match(urls?.url!, /^\/uploads\/original.+\?signature=.+$/)

        ctx.response.send(responsiveAttachment)
        ctx.response.finish()
      })
    })

    await supertest(server)
```

```
      .post('/')
      .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg'))
  })
})

test.group('Error checks', (group) => {
  group.before(async () => {
    app = await setupApplication()
    await setup(app)

    app.container.resolveBinding('Adonis/Core/Route').commit()
    ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
  })

  group.after(async () => {
    await cleanup(app)
  })
})

test.group('ResponsiveAttachment | Custom breakpoints', (group) => {
  group.before(async () => {
    app = await setupApplication()
    await setup(app)

    app.container.resolveBinding('Adonis/Core/Route').commit()
    ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
  })

  group.after(async () => {
    await cleanup(app)
  })

  test('create attachment from the user uploaded image', async (assert) => {
    const Drive = app.container.resolveBinding('Adonis/Core/Drive')

    const server = createServer((req, res) => {
      const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)
      app.container.make(BodyParserMiddleware).handle(ctx, async () => {
        const file = ctx.request.file('avatar')!
        const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
        responsiveAttachment?.setOptions({
          breakpoints: { small: 400, medium: 700, large: 1000, xlarge: 1200 },
        })
        await responsiveAttachment?.save()

        assert.isTrue(responsiveAttachment?.isPersisted)
        assert.isTrue(responsiveAttachment?.isLocal)

        assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
        assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.thumbnail.name!))
        assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.small.name!))
        assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.medium.name!))
```

```
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.large.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.xlarge.name!))

      ctx.response.send(responsiveAttachment)
      ctx.response.finish()
    })
  })

  const { body } = await supertest(server)
    .post('/')
    .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg'))

  assert.isTrue(await Drive.exists(body.name))
  assert.isTrue(await Drive.exists(body.breakpoints?.thumbnail.name))
  assert.isTrue(await Drive.exists(body.breakpoints?.small.name))
  assert.isTrue(await Drive.exists(body.breakpoints?.medium.name))
  assert.isTrue(await Drive.exists(body.breakpoints?.large.name))
  assert.isTrue(await Drive.exists(body.breakpoints?.xlarge.name))
})

test('pre-compute urls for newly created images', async (assert) => {
  const Drive = app.container.resolveBinding('Adonis/Core/Drive')

  const server = createServer((req, res) => {
    const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)

    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const file = ctx.request.file('avatar')!
      const responsiveAttachment = await ResponsiveAttachment.fromFile(file)
      responsiveAttachment?.setOptions({
        breakpoints: { small: 400, medium: 700, large: 1000, xlarge: 1200 },
        preComputeUrls: true,
      })
      await responsiveAttachment?.save()

      assert.isTrue(responsiveAttachment?.isPersisted)
      assert.isTrue(responsiveAttachment?.isLocal)

      assert.isNotEmpty(responsiveAttachment?.urls)

      assert.isDefined(responsiveAttachment?.url)
      assert.isDefined(responsiveAttachment?.urls?.breakpoints?.xlarge.url)
      assert.isDefined(responsiveAttachment?.urls?.breakpoints?.large.url)
      assert.isDefined(responsiveAttachment?.urls?.breakpoints?.medium.url)
      assert.isDefined(responsiveAttachment?.urls?.breakpoints?.small.url)
      assert.isDefined(responsiveAttachment?.urls?.breakpoints?.thumbnail.url)

      assert.isNotNull(responsiveAttachment?.url)
      assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.xlarge.url)
      assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.large.url)
      assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.medium.url)
      assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.small.url)
```

```
      assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.thumbnail.url)

      assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.xlarge.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.large.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.medium.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.small.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.thumbnail.name!))

      assert.equal(responsiveAttachment?.width, 1500)
      assert.equal(responsiveAttachment?.breakpoints?.xlarge.width!, 1200)
      assert.equal(responsiveAttachment?.breakpoints?.large.width!, 1000)
      assert.equal(responsiveAttachment?.breakpoints?.medium.width!, 700)
      assert.equal(responsiveAttachment?.breakpoints?.small.width!, 400)

      assert.isTrue(
        responsiveAttachment?.breakpoints?.thumbnail.size! <
          responsiveAttachment?.breakpoints!.small.size!
      )
      assert.isTrue(
        responsiveAttachment?.breakpoints?.small.size! <
          responsiveAttachment?.breakpoints!.medium.size!
      )
      assert.isTrue(
        responsiveAttachment?.breakpoints?.medium.size! <
          responsiveAttachment?.breakpoints!.large.size!
      )
      assert.isTrue(
        responsiveAttachment?.breakpoints?.large.size! <
          responsiveAttachment?.breakpoints?.xlarge.size!
      )
      assert.isTrue(responsiveAttachment?.breakpoints!.xlarge.size! < responsiveAttachment?.size!)

      ctx.response.send(responsiveAttachment)
      ctx.response.finish()
    })
  })

  await supertest(server)
    .post('/')
    .attach('avatar', join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg'))
 })
})

test.group('ResponsiveAttachment | fromBuffer', (group) => {
 group.before(async () => {
  app = await setupApplication()
  await setup(app)

  app.container.resolveBinding('Adonis/Core/Route').commit()
  ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
 })
```

```
group.after(async () => {
  await cleanup(app)
})

test('create attachment from the user-provided buffer', async (assert) => {
  const Drive = app.container.resolveBinding('Adonis/Core/Drive')

  const server = createServer((req, res) => {
    const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)
    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const readableStream = await readFile(
        join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg')
      )
      const responsiveAttachment = await ResponsiveAttachment.fromBuffer(readableStream)
      await responsiveAttachment.save()

      assert.isTrue(responsiveAttachment.isPersisted)
      assert.isTrue(responsiveAttachment.isLocal)

      assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.thumbnail.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.small.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.medium.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.large.name!))

      ctx.response.send(responsiveAttachment)
      ctx.response.finish()
    })
  })

  const { body } = await supertest(server).post('/')

  assert.isTrue(await Drive.exists(body.name))
})

test('pre-compute url for newly-created images', async (assert) => {
  const Drive = app.container.resolveBinding('Adonis/Core/Drive')

  const server = createServer((req, res) => {
    const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)

    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const readableStream = await readFile(
        join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg')
      )
      const responsiveAttachment = await ResponsiveAttachment.fromBuffer(readableStream)
      responsiveAttachment.setOptions({ preComputeUrls: true })
      await responsiveAttachment.save()

      assert.isTrue(responsiveAttachment.isPersisted)
      assert.isTrue(responsiveAttachment.isLocal)
```

```
    assert.isNotEmpty(responsiveAttachment?.urls)

    assert.isDefined(responsiveAttachment?.url)
    assert.isDefined(responsiveAttachment?.urls?.breakpoints?.large.url)
    assert.isDefined(responsiveAttachment?.urls?.breakpoints?.medium.url)
    assert.isDefined(responsiveAttachment?.urls?.breakpoints?.small.url)
    assert.isDefined(responsiveAttachment?.urls?.breakpoints?.thumbnail.url)

    assert.isNotNull(responsiveAttachment?.url)
    assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.large.url)
    assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.medium.url)
    assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.small.url)
    assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.thumbnail.url)

    assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
    assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.large.name!))
    assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.medium.name!))
    assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.small.name!))
    assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.thumbnail.name!))

    assert.isTrue(
      responsiveAttachment?.breakpoints!.thumbnail.size! <
        responsiveAttachment?.breakpoints!.small.size!
    )
    assert.isTrue(
      responsiveAttachment?.breakpoints!.small.size! <
        responsiveAttachment?.breakpoints!.medium.size!
    )
    assert.isTrue(
      responsiveAttachment?.breakpoints!.medium.size! <
        responsiveAttachment?.breakpoints!.large.size!
    )
    assert.isTrue(responsiveAttachment?.breakpoints!.large.size! < responsiveAttachment?.size!)

    ctx.response.send(responsiveAttachment)
    ctx.response.finish()
  })
 })

 const { body } = await supertest(server).post('/')

 assert.isDefined(body.url)
})

test('delete local images', async (assert) => {
 const Drive = app.container.resolveBinding('Adonis/Core/Drive')

 const server = createServer((req, res) => {
  const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)

  app.container.make(BodyParserMiddleware).handle(ctx, async () => {
```

```
      const readableStream = await readFile(
        join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg')
      )
      const responsiveAttachment = await ResponsiveAttachment.fromBuffer(readableStream)
      responsiveAttachment.setOptions({ preComputeUrls: true })
      await responsiveAttachment.save()
      await responsiveAttachment.delete()

      assert.isFalse(responsiveAttachment?.isPersisted)
      assert.isTrue(responsiveAttachment?.isLocal)
      assert.isTrue(responsiveAttachment?.isDeleted)

      ctx.response.send(responsiveAttachment)
      ctx.response.finish()
    })
  })

  const { body } = await supertest(server).post('/')

  assert.isDefined(body.url)
  assert.isDefined(body.breakpoints.large.url)
  assert.isDefined(body.breakpoints.medium.url)
  assert.isDefined(body.breakpoints.small.url)
  assert.isDefined(body.breakpoints.thumbnail.url)

  assert.isNotNull(body.url)
  assert.isNotNull(body.breakpoints.large.url)
  assert.isNotNull(body.breakpoints.medium.url)
  assert.isNotNull(body.breakpoints.small.url)
  assert.isNotNull(body.breakpoints.thumbnail.url)

  assert.isFalse(await Drive.exists(body.name))
  assert.isFalse(await Drive.exists(body.breakpoints.large.name))
  assert.isFalse(await Drive.exists(body.breakpoints.medium.name))
  assert.isFalse(await Drive.exists(body.breakpoints.small.name))
  assert.isFalse(await Drive.exists(body.breakpoints.thumbnail.name))
 })
})

test.group('ResponsiveAttachment | miscellaneous', (group) => {
 group.before(async () => {
  app = await setupApplication()
  await setup(app)

  app.container.resolveBinding('Adonis/Core/Route').commit()
  ResponsiveAttachment.setDrive(app.container.resolveBinding('Adonis/Core/Drive'))
 })

 group.after(async () => {
  await cleanup(app)
 })
```

```
test('throw error if unallowed file type is provided to `fromBuffer` method', async (assert) => {
  const Drive = app.container.resolveBinding('Adonis/Core/Drive')

  const server = createServer((req, res) => {
    const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)
    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const readableStream = await readFile(
        join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg')
      )
      const responsiveAttachment = await ResponsiveAttachment.fromBuffer(readableStream)
      await responsiveAttachment.save()

      assert.isTrue(responsiveAttachment.isPersisted)
      assert.isTrue(responsiveAttachment.isLocal)

      assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.thumbnail.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.small.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.medium.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.large.name!))

      ctx.response.send(responsiveAttachment)
      ctx.response.finish()
    })
  })

  const { body } = await supertest(server).post('/')

  assert.isTrue(await Drive.exists(body.name))
})

test('pre-compute url for newly-created images', async (assert) => {
  const Drive = app.container.resolveBinding('Adonis/Core/Drive')

  const server = createServer((req, res) => {
    const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)

    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const readableStream = await readFile(
        join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg')
      )
      const responsiveAttachment = await ResponsiveAttachment.fromBuffer(readableStream)
      responsiveAttachment.setOptions({ preComputeUrls: true })
      await responsiveAttachment.save()

      assert.isTrue(responsiveAttachment.isPersisted)
      assert.isTrue(responsiveAttachment.isLocal)

      assert.isNotEmpty(responsiveAttachment?.urls)

      assert.isDefined(responsiveAttachment?.url)
      assert.isDefined(responsiveAttachment?.urls?.breakpoints?.large.url)
```

```
      assert.isDefined(responsiveAttachment?.urls?.breakpoints?.medium.url)
      assert.isDefined(responsiveAttachment?.urls?.breakpoints?.small.url)
      assert.isDefined(responsiveAttachment?.urls?.breakpoints?.thumbnail.url)

      assert.isNotNull(responsiveAttachment?.url)
      assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.large.url)
      assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.medium.url)
      assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.small.url)
      assert.isNotNull(responsiveAttachment?.urls?.breakpoints?.thumbnail.url)

      assert.isTrue(await Drive.exists(responsiveAttachment?.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.large.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.medium.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.small.name!))
      assert.isTrue(await Drive.exists(responsiveAttachment?.breakpoints?.thumbnail.name!))

      assert.isTrue(
        responsiveAttachment?.breakpoints!.thumbnail.size! <
          responsiveAttachment?.breakpoints!.small.size!
      )
      assert.isTrue(
        responsiveAttachment?.breakpoints!.small.size! <
          responsiveAttachment?.breakpoints!.medium.size!
      )
      assert.isTrue(
        responsiveAttachment?.breakpoints!.medium.size! <
          responsiveAttachment?.breakpoints!.large.size!
      )
      assert.isTrue(responsiveAttachment?.breakpoints!.large.size! < responsiveAttachment?.size!)

      ctx.response.send(responsiveAttachment)
      ctx.response.finish()
    })
  })

  const { body } = await supertest(server).post('/')

  assert.isDefined(body.url)
})

test('delete local images', async (assert) => {
  const Drive = app.container.resolveBinding('Adonis/Core/Drive')

  const server = createServer((req, res) => {
    const ctx = app.container.resolveBinding('Adonis/Core/HttpContext').create('/', {}, req, res)

    app.container.make(BodyParserMiddleware).handle(ctx, async () => {
      const readableStream = await readFile(
        join(__dirname, '../Statue-of-Sardar-Vallabhbhai-Patel-1500x1000.jpg')
      )
      const responsiveAttachment = await ResponsiveAttachment.fromBuffer(readableStream)
      responsiveAttachment.setOptions({ preComputeUrls: true })
```

```
      await responsiveAttachment.save()
      await responsiveAttachment.delete()

      assert.isFalse(responsiveAttachment?.isPersisted)
      assert.isTrue(responsiveAttachment?.isLocal)
      assert.isTrue(responsiveAttachment?.isDeleted)

      ctx.response.send(responsiveAttachment)
      ctx.response.finish()
    })
  })

  const { body } = await supertest(server).post('/')

  assert.isDefined(body.url)
  assert.isDefined(body.breakpoints.large.url)
  assert.isDefined(body.breakpoints.medium.url)
  assert.isDefined(body.breakpoints.small.url)
  assert.isDefined(body.breakpoints.thumbnail.url)

  assert.isNotNull(body.url)
  assert.isNotNull(body.breakpoints.large.url)
  assert.isNotNull(body.breakpoints.medium.url)
  assert.isNotNull(body.breakpoints.small.url)
  assert.isNotNull(body.breakpoints.thumbnail.url)

  assert.isFalse(await Drive.exists(body.name))
  assert.isFalse(await Drive.exists(body.breakpoints.large.name))
  assert.isFalse(await Drive.exists(body.breakpoints.medium.name))
  assert.isFalse(await Drive.exists(body.breakpoints.small.name))
  assert.isFalse(await Drive.exists(body.breakpoints.thumbnail.name))
 })
})
```