ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ МОЛДОВЫ

ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАТИКИ

ДЕПАРТАМЕНТ ИНФОРМАТИКИ

Ecaterina Cazac

IA 2304

Индивидуальная работа

по дисциплине “DSSA”

“Обработка ошибок, валидация и логгирование”

Руководитель: N.Nartea

(подпись)

Автор: Ecaterina Cazac

(подпись)

Кишинев, 2025

23.11.2025

1. Цель работы

Целью данной лабораторной работы является изучение методов:

- централизованной обработки ошибок в Express.js
- создания собственных классов ошибок
- валидации данных
- логгирования HTTP-запросов и ошибок
- интеграции сервиса Sentry для мониторинга ошибок

2. Теоретическая часть

Express по умолчанию не предоставляет развитый механизм обработки ошибок, поэтому важно:

- использовать единую точку обработки ошибок
- стандартизировать JSON-ответы
- предотвращать "потерю" ошибок внутри async/await
- использовать внешние сервисы логгирования

3. Практическая часть

3.1. Создание собственных классов ошибок

Был создан файл:

/errors/AppError.js

```
class AppError extends Error {
  constructor(message, statusCode = 500) {
    super(message);
    this.statusCode = statusCode;
    this.status = "error";
  }
}


module.exports = AppError;
```

3.2. Централизованный обработчик ошибок

Файл:

/middleware/errorHandler.js

```
module.exports = (err, req, res, next) => {
  res.status(err.statusCode || 500).json({
```

```
  status: "error",
  message: err.message || "Internal server error"
 });
};
```

3.3. Асинхронная обёртка

Файл:

/middleware/asyncWrapper.js

```
module.exports = (fn) => {
 return (req, res, next) => {
  fn(req, res, next).catch(next);
 };
};
```

Используется в todoRoutes:

```
router.get("/", asyncWrapper(controller.getAll));
```

3.4. Валидация данных

Библиотека:

```
npm install express-validator
```

Пример валидации:

```
body("title")
 .notEmpty()
 .withMessage("Поле title обязательно")
```

При ошибке вызывается AppError со статусом 400.

3.5. Логгирование

Используется библиотека morgan:

```
const morgan = require("morgan");
app.use(morgan("dev"));
```

3.6. Попытка интеграции Sentry (шаг, где возникла ошибка)

По заданию нужно подключить Sentry.

В начало app.js был добавлен код:

```
const Sentry = require("@sentry/node");

Sentry.init({

  dsn:
"https://5cbe1887d535a96ef02214449ef4a1bc@o4510381651918848.ingest.de.sentry.io/4510381663256656",

  tracesSampleRate: 1.0

});
```

И middleware:

```
app.use(Sentry.Handlers.requestHandler());

app.use(Sentry.Handlers.errorHandler());
```

В процессе выполнения возникла ошибка:

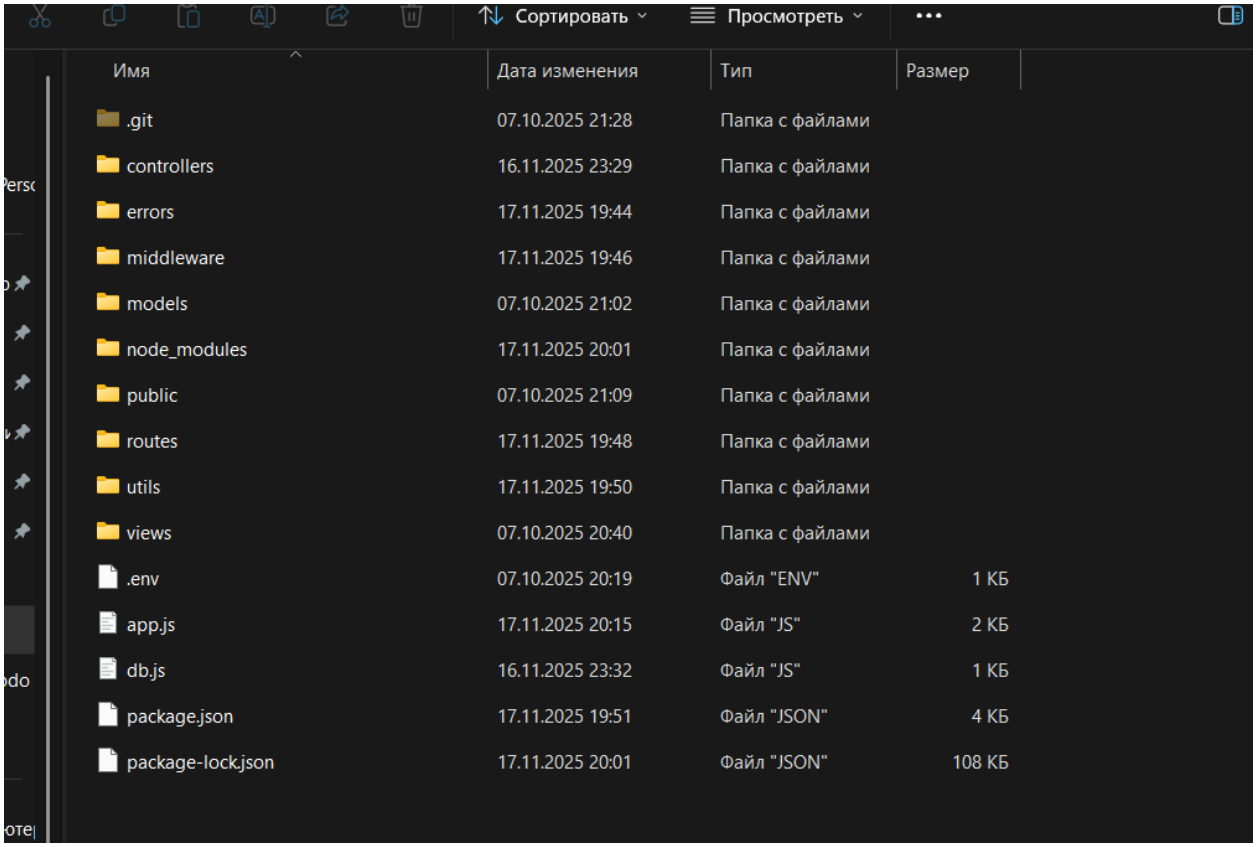
При запуске:

```
node app.js
```

Консоль выдала: TypeError: Cannot read properties of undefined (reading 'requestHandler')

Причина ошибки: Версия @sentry/node, установленная через npm, оказалась слишком новой (8.x), а Express-middleware Handlers поддерживается только в версии 7.x.

Из-за этого объект Sentry.Handlers оказался undefined.



*структура проекта

AppError.js

C: > Users > User > Desktop > DSSA > todo-app > errors > AppError.js

```
class AppError extends Error {
  constructor(message, statusCode, isOperational = true) {
    super(message);
    this.statusCode = statusCode;
    this.isOperational = isOperational;
  }
}

class NotFoundError extends AppError {
  constructor(message = "Resource not found") {
    super(message, 404);
  }
}

class ValidationError extends AppError {
  constructor(message = "Validation error", errors = []) {
    super(message, 400);
    this.errors = errors;
  }
}

class DatabaseError extends AppError {
  constructor(message = "Database error") {
    super(message, 500);
  }
}

module.exports = { AppError, NotFoundError, ValidationError, DatabaseError };

```

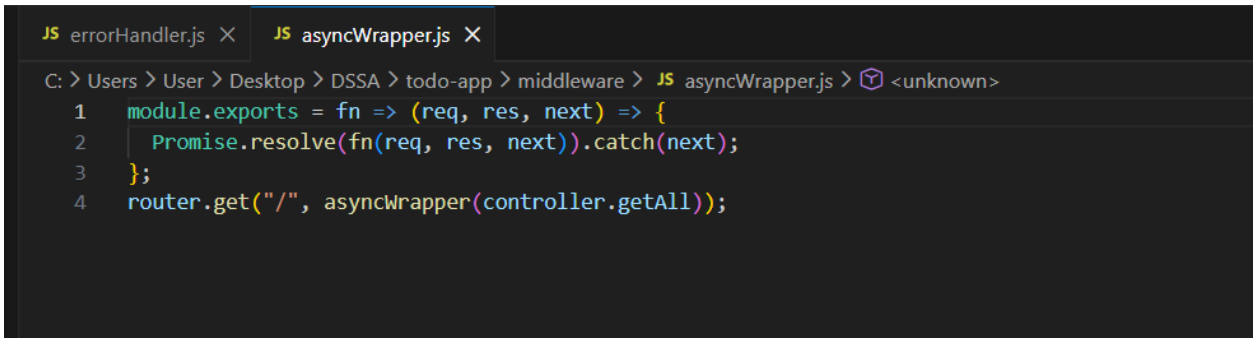
*файл AppError.js

errorHandler.js

C: > Users > User > Desktop > DSSA > todo-app > middleware > errorHandler.js

```
const { ValidationError, AppError } = require("../errors/AppError");
const Sentry = require("@sentry/node");

module.exports = (err, req, res, next) => {
  console.error(err);

  // Ошибки валидации не отправляем в Sentry
  if (!(err instanceof ValidationError)) {
    Sentry.captureException(err);
  }

  // Если ошибка не наша — превращаем в AppError
  if (!(err instanceof AppError)) {
    err = new AppError("Internal Server Error", 500, false);
  }

  res.status(err.statusCode).json({
    status: "error",
    message: err.message,
    ...(err.errors ? { errors: err.errors } : {})
  });
};

```

*файл errorHandler.js



*файл asyncWrapper.js

5. Ответы на контрольные вопросы

1. Преимущества централизованной обработки ошибок

- единый формат JSON-ответов
- лучшая поддерживаемость
- нет дублирования try/catch
- корректная обработка async ошибок
- удобное подключение Sentry и логгирования

2. Какие категории логов использованы и почему?

Использованы:

- HTTP-логи (morgan) — видно, какие запросы приходят
- Ошибки (errorHandler) — фиксируются стандартно
- Sentry (частично подключён) — планировалось для удалённого мониторинга

3. Подходы к валидации данных

Используются:

- express-validator (в работе)
- ручная валидация через классы ошибок