JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 9, No. 2, Juni 2024, Pp. 568-577

PENERAPAN *TEST DRIVEN DEVELOPMENT* DALAM PENGEMBANGAN *BACKEND* UNTUK KEBUTUHAN DATA PADA APLIKASI *PEER-TO-PEER LENDING SYARIAH*

Farhan Hilmi*[1)], Rahmat Fauzi[2)], Ekky Novriza Alam[3)]
1. Universitas Telkom, Indonesia
2. Universitas Telkom, Indonesia
3. Universitas Telkom, Indonesia

Article Info
Kata Kunci: *API; Fintech; P2P; TDD*

Keywords: *API; Fintech; P2P; TDD*

Article history:
Received 13 February 2024
Revised 27 February 2024
Accepted 12 March 2024
Available online 1 June 2024

DOI :
https://doi.org/10.29100/jipi.v9i2.4631

* Corresponding author.
Farhan Hilmi
E-mail address:
farhanhilmi@student.telkomuniversity.ac.id

ABSTRAK

Kemajuan transformasi digital telah mengubah berbagai aktivitas, termasuk transaksi keuangan. *Fintech* (*financial technology*) menjadi penggerak penting dalam layanan keuangan, termasuk *peer-to-peer* (*P2P*) lending. Namun, model *fintech* konvensional memiliki tantangan seperti bunga fluktuatif tinggi dan pelanggaran data. Solusi hadir dalam bentuk *fintech syariah*, mengutamakan prinsip-prinsip Islam dan inklusi keuangan. Namun, munculnya platform *fintech* ilegal menyebabkan eksploitasi konsumen dan keluhan hukum. Studi ini mengusulkan implementasi *Test Driven Development* (*TDD*) untuk menciptakan *backend Application Programming Interface* (*API*) *P2P* lending yang aman dan efisien pada *fintech* berbasis *syariah*. *TDD* memungkinkan pengembang merancang dan mengembangkan *API* sesuai dengan tes yang telah ditentukan. *API* mengikuti prinsip arsitektur *Representational State Transfer* (*REST*). Penelitian ini melibatkan pengujian *integrations API* dan *load testing* untuk mengevaluasi kinerjanya dalam berbagai skenario pengguna. Hasil menunjukkan bahwa metode yang digunakan efektif dalam mengurangi bugs yang ada pada *API* karena dapat langsung terdeteksi pada saat proses pengembangan. Selain itu, berdasarkan hasil load testing yang dilakukan dengan jumlah pengguna 50 sampai dengan 300 pengguna yang melakukan *hit API* secara bersamaan tidak menyebabkan *error* dan *response time* rata – rata masih berkisaran pada angka yang dapat diterima oleh pengguna tanpa merasakan adanya gangguan yaitu berkisar 138 *milisecond* sampai 1420 *milisecond*.

ABSTRACT

The advancement of digital transformation has revolutionized various activities, including financial transactions. Fintech (financial technology) has become a crucial driver in financial services, including peer-to-peer (P2P) lending. However, conventional fintech models face challenges such as high fluctuating interest rates and data breaches. The solution comes in the form of Shariah-compliant fintech, prioritizing Islamic principles and financial inclusion. Nevertheless, the emergence of illegal fintech platforms has led to consumer exploitation and legal complaints. This study proposes the implementation of Test-Driven Development (TDD) to create a secure and efficient backend Application Programming Interface (API) for Shariah-based P2P lending fintech. TDD enables developers to design and develop APIs based on predetermined tests. The API adheres to the principles of Representational State Transfer (REST) architecture. The research involves API integration testing and load testing to evaluate its performance under various user scenarios. The results demonstrate that the method used effectively reduces existing bugs in the API as they can be promptly detected during the development process. Furthermore, based on the load testing results with 50 to 300 simultaneous users hitting the API, no errors were observed, and the average response time remained within an acceptable range for users without experiencing any disruptions, ranging from 138 milliseconds to 1420 milliseconds.

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 9, No. 2, Juni 2024, Pp. 568-577

I. PENDAHULUAN

PERKEMBANGAN Perkembangan transformasi digital telah mengakibatkan banyak perubahan yang mempermudah berbagai aktivitas. Saat ini, internet dan teknologi informasi menjadi faktor penunjang yang krusial dalam menjalankan proses bisnis perusahaan. Begitu pula dalam bidang finansial, di mana transformasi digital telah mempengaruhi cara melakukan transaksi keuangan seperti pembayaran, pinjaman *online*, penggalangan dana, *microfinancing*, investasi, dan *P2P lending* [1]. Penggunaan teknologi informasi dalam konteks finansial ini dikenal sebagai *financial technology* (*fintech*), yaitu inovasi layanan keuangan yang memanfaatkan teknologi informasi untuk meningkatkan efektivitas dan efisiensi dalam proses transaksi keuangan [2].

Salah satu platform yang ditawarkan oleh jasa *fintech* yaitu *peer to peer lending* yang dapat memungkinkan pihak peminjam yang membutuhkan dana untuk mendapatkan pendanaan dari pihak *lender* yang bertindak sebagai investor [3]. Menurut data yang berasal dari OJK per agustus 2022 terdapat 14,328,221 akun penerima. Jumlah akun peminjam mengalami kenaikan dibandingkan tahun 2019 pada periode yang sama dengan total sebesar 12,832,271.

Hal tersebut membuktikan bahwa bisnis industri *fintech P2P lending* semakin berkembang dalam beberapa tahun terakhir. *Fintech* konvensional saat ini mayoritas mengandung riba berupa bunga tinggi yang bersifat fluktuatif, waktu pengembalian pendanaan yang memiliki batasan sampai 12 bulan serta risiko bocornya data nasabah [4]. Untuk itu, *fintech* berbasis *syariah* kini hadir dan mengalami perkembangan dalam hal penggunaan. *Fintech syariah* [5] merupakan suatu teknologi keuangan yang memudahkan dalam melakukan transaksi yang berpedoman pada nilai – nilai *syariah*. *Fintech syariah* memiliki beberapa karakteristik untuk menjaga proses bisnisnya tidak keluar dari *syariah*, yaitu tidak mengandung unsur *riba, ghoror, mudharat, jahalal*. Industri *fintech syariah* memiliki peluang yang besar untuk meningkatkan inklusi keuangan *syariah* yang berada pada angka 9,1%, angka tersebut sangat berbeda jauh jika dibandingkan dengan inklusi keuangan *fintech* konvensional yaitu sebesar 76,19% [6].

Namun seiring dengan maraknya pinjaman *online* yang beredar, hingga saat ini masih terdapat penyalahgunaan yang memanfaatkan celah dalam teknologi keuangan. Banyak kasus yang terjadi dimana perusahaan *fintech* tidak memiliki izin untuk beroprasi atau ilegal, hal itu dapat merugikan pengguna yang melakukan peminjaman lewat platform ilegal tersebut [7]. Platform *fintech* ilegal memiliki beberapa ciri [8] yaitu tidak terdapat logo OJK dalam aplikasi atau website, tidak terdaftar pada Google Play Store, tingkat bunga yang tinggi, serta denda dan biaya layanan yang sangat tinggi, proses pencairan pada platform ilegal biasanya sangat mudah, alamat kantor yang tidak jelas dan susah untuk dihubungi, proses penagihan pinjaman yang tidak etis. Pada tahun 2019 sampai tahun 2021 yang dilakukan oleh pihak perusahaan pinjaman *online* konvensional, pelanggaran tersebut berdasarkan laporan yang diajukan oleh masyarakat. Pada periode tersebut pelanggaran ringan yang terjadi sebanyak 10.441, sedangkan untuk pelanggaran berat sebesar 9.270 pelanggaran yang terjadi. Faktor yang menyebabkan ketertarikan masyarakan pada pinjaman online tersebut yaitu proses melakukan permintaan dana dilakukan dengan mudah dan cepat, apalagi dalam situasi mendesak dimana uang sangat dibutuhkan sehingga masyarakat memilih untuk menggunakan layanan pinjaman *online* [9].

Untuk mengatasi masalah di atas, maka diperlukan suatu aplikasi yang sesuai dengan pedoman pinjaman *online*, maka pengembangan pada *backend* harus sesuai dengan kebutuhan yang diperlukan dan tidak terdapat kesalahan pada *business logic* yang diterapkan. Pada penelitian yang terdahulu [10] menyebutkan dalam hasil penelitian tersebut bahwa pengembangan aplikasi dengan menggunakan metode *Test Driven Development* dapat memangkas *bugs* yang ada sehingga berkontribusi pada peningkatan kualitas perangkat lunak serta meningkatkan kepuasan pelanggan dan memiliki potensi untuk secara signifikan mengurangi kerapatan cacat pada perangkat lunak yang dikembangkan, baik secara instan maupun dalam jangka waktu yang lebih panjang. Selaint itu, pada penelitian terdahulu juga menyebutkan *TDD* memiliki peran penting dalam mendeteksi perubahan yang berpotensi melanggar *requirement* yang telah ditentukan. Penelitian tersebut berfokus pada pengembangan aplikasi yang terhindar dari kecacatan pada fitur yang dikembangkan, namun *response time* atau waktu yang diperlukan untuk menyampaikan data ke pengguna tidak dilakukan analisis. Sehingga memungkinkan aplikasi terhindar dari *bugs* namun terjadi *delay* atau keterlambatan dalam mengakases suatu data.

Penelitian ini bertujuan untuk mengembangkan *Application Programming Interface* yang terhindar dari berbagai kecacatan serta memiliki *response time* yang tidak membuat pelanggan merasakan keterlambatan dalam menerima konten atau data sehingga dibutuhkan *load testing* untuk memastikan hal tersebut. Dalam penelitian ini peran *backend* untuk merancang dan mengembangkan *backend Application Programming Interface* (*API*) [11] yang berguna untuk menyediakan kebutuhan data yang diperlukan dengan menggunakan pendekatan metode yang digunakan yaitu *Test Driven Development* (*TDD*). Setelah *test case* dibuat, *developer* akan memulai melakukan

coding untuk memenuhi *test case* sampai tidak ada kesalahan atau kegagalan pada *testing* yang dilakukan. Dalam penelitian yang dilakukan oleh [12] menyebutkan bahwa *Test Driven Development* merupakan salah satu metode yang sering digunakan untuk mengembangkan perangkat lunak yang ide intinya yaitu merancang perangkat lunak secara bertahap dengan melakukan *testing* yang akan mempermudah proses pengembangan. Tahapan yang dilakukan pada metode ini antara lain *write a test case, write code, refactor, iterate*. Selain itu, untuk memastikan *API* yang dikembangkan tidak mengalami kelambatan dalam melakukan komunikasi antar *server* dan *client*, maka harus dilakukan *load testing* untuk mengetahui *response time* yang didapatkan pada berbagai kasus dengan jumlah pengguna yang berbeda pada setiap kali pengujian dilakukan.

II. METODE PENELITIAN

Dalam penelitian ini, kami menyusun sebuah kerangka penelitian yang komprehensif untuk mengatasi permasalahan yang dihadapi. Pendekatan kami melibatkan serangkaian tahapan yang terstruktur dengan dimulai mengkaji literatur yang ada untuk memahami secara mendalam domain masalah dan mengidentifikasi tantangan serta solusi yang potensial. Secara bersamaan, kami mengumpulkan data relevan dari sumber-sumber terpercaya untuk membimbing proses pengembangan dan memastikan bahwa *API* yang kami bangun sesuai dengan kebutuhan pengguna.

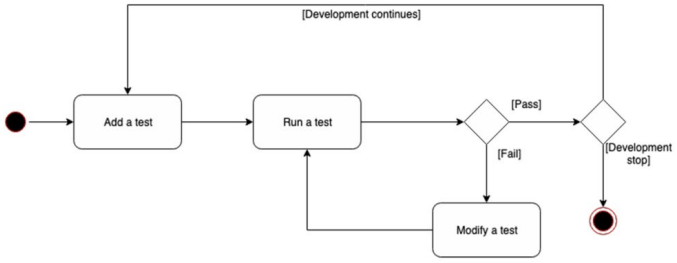
Setelah fase pengumpulan informasi, kami mulai mengembangkan aplikasi sebenarnya, dengan tetap berpegang teguh pada prinsip-prinsip *test driven development*. *TDD* memungkinkan tim pengembanguntuk menciptakan *API* secara bertahap dan berulang [13], dengan terus memverifikasi fungsionalitasnya melalui serangkaian pengujian. Pendekatan ini memungkinkan kami mendeteksi dan memperbaiki potensi masalah sejak awal, sehingga hasil akhirnya menjadi lebih kokoh dan handal. Dalam penelitian ini Gambar. 1, peneliti melakukan studi literatur yang bersumber dari berbagai buku, jurnal, dan *conference* dalam rangka meningkatkan pemahaman mengenai solusi yang dapat diberikan pada permasalahan yang ada serta mencari metode yang tepat dalam menyelesaikan permasalahan tersebut.

Gambar. 1. Kerangka penelitian

A. *Test Driven Development* (*TDD*)

Test Driven Development (*TDD*) [12] adalah suatu pendekatan dalam pengembangan perangkat lunak di mana tujuan dari program yang akan dibuat ditentukan terlebih dahulu. Setelah itu, beberapa skenario pengujian dibuat untuk menguji kasus-kasus yang berpotensi menyebabkan kesalahan dalam program. Pada metode pengembangan ini, jenis pengujian yang digunakan adalah *integrations testing* pada *endpoint*. Setelah tes dibuat, langkah berikutnya adalah menulis kode program hingga tidak ada lagi kesalahan yang terdeteksi dalam pengujian. Pendekatan ini bertujuan untuk mengurangi kemungkinan kesalahan yang mungkin terjadi dalam program yang dikembangkan [14]. Pada Gambar. 2 dijelaskan mengenai tahap – tahap dalam melakukan *TDD* saat mengembangkan aplikasi.

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 9, No. 2, Juni 2024, Pp. 568-577



Gambar. 2. Proses *test driven development*

Dalam mengembangkan perangkat lunak menggunakan metode *TDD*, terdapat serangkaian tahapan yang perlu diikuti. Tahap awal dimulai dengan menetapkan tujuan untuk fitur yang akan dikembangkan dan menerapkannya dalam bentuk pengujian. Selanjutnya, jalankan tes untuk menguji apakah fitur tersebut telah berfungsi dengan benar atau masih mengalami kesalahan. Jika masih terdapat kesalahan, langkah selanjutnya adalah melakukan perubahan pada kode program guna memperbaiki kesalahan tersebut. Jika pengujian telah berhasil dan fitur telah lolos uji, maka tahap pengembangan fitur dianggap selesai, dan siklus ini akan terus diulang hingga seluruh fitur yang dibutuhkan dalam aplikasi telah dikembangkan dengan baik.

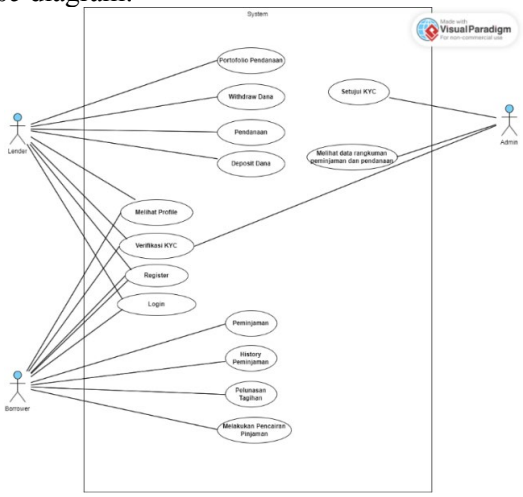
B. *REST API*

Representational State Transfer (*REST*) adalah suatu pendekatan arsitektur yang berguna dalam pembuatan dan pengelolaan sistem yang terdistribusi [11]. *REST* terdiri dari sejumlah aturan yang harus diikuti sesuai dengan prinsip arsitektur untuk menciptakan sistem dengan desain *interface* yang baik dan mudah dipahami. Dalam implementasi arsitektur *REST API*, terdapat beberapa *constraint* [15] yang harus dipatuhi, antara lain *client-server* yaitu *separation of concerns* dimana *client* tidak harus mengetahui *business logic* atau proses yang terjadi pada *server* dan *server* tidak harus mengetahui apapun mengenai *user interface* yang ada pada *client,* lalu terdapat *stateless* yaitu komunikasi antar *client* dan *server* harus tidak saling terikat (*stateless*), artinya setiap *request* yang dilakukan oleh *client* harus memenuhi semua informasi yang dibutuhkan *server*, sehingga *server* tidak perlu menyimpan data kedalam *session* , sedangkan *cacheable* yaitu untuk menyimpan data yang telah didapatkan dari *server* untuk waktu yang telah ditentukan. *Contrains* lainnya yaitu *uniform interface, layered system,* dan *code-on-demand*.

III. HASIL DAN PEMBAHASAN

A. *Use Case Diagram*

Use case diagram ini menjadi alat yang efektif untuk memahami bagaimana berbagai elemen dalam sistem saling berinteraksi dan berperan dalam menjalankan proses-proses yang diinginkan. Dengan adanya *use case* diagram, dapat lebih mudah mengidentifikasi dan menggambarkan skenario-skenario yang berhubungan dengan penggunaan sistem, sehingga memudahkan dalam perancangan dan pengembangan aplikasi yang lebih baik [16]. Gambar. 3 menunjukan hasil rancangan *use case* diagram.



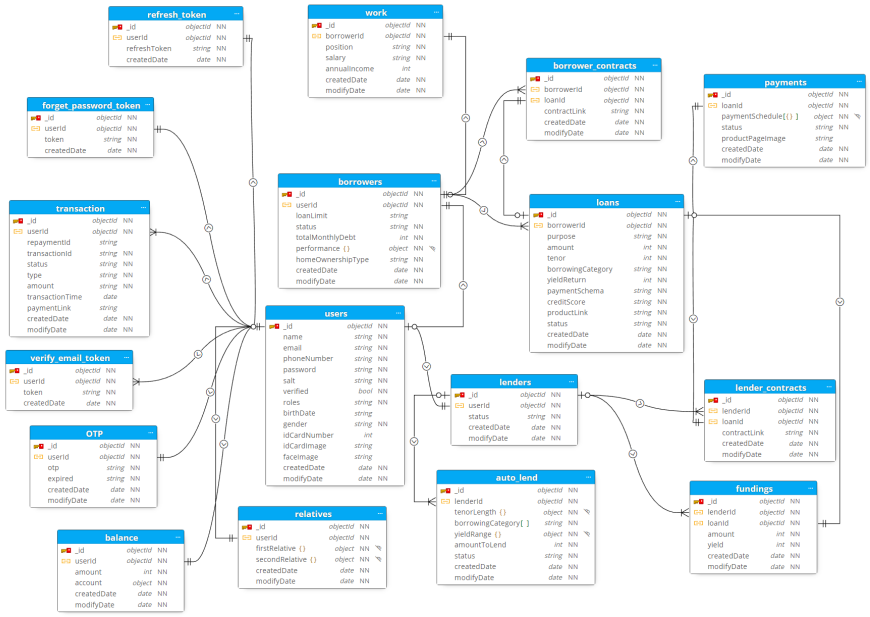
Gambar. 3. Rancangan *use case* diagram

B. *Entity Relationship Diagram*

Entity Relationship Diagram digunakan untuk menggambarkan struktur statis dari suatu sistem perangkat lunak.

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 9, No. 2, Juni 2024, Pp. 568-577

Diagram ini memvisualisasikan kelas-kelas, atribut, dan hubungan antara kelas-kelas yang membentuk bagian dari sistem tersebut [16]. Terdapat beberapa entitas diataranya yaitu entitas lender yang menyimpan data – data terkait dengan informasi akun pendana, selain itu, terdapat entitas *borrower*. Kedua entitas tersebut terhubungan dengan entitas user yang menyimpan informasi data diri dari pengguna aplikasi. Entitas lainnya yaitu *fundings* yang berguna untuk menyimpan data pendanaan yang dilakukan oleh *lender* dan *loans* untuk menyimpan permintaan peminjaman dana yang dilakukan oleh *borrower* seperti pada Gambar. 4.

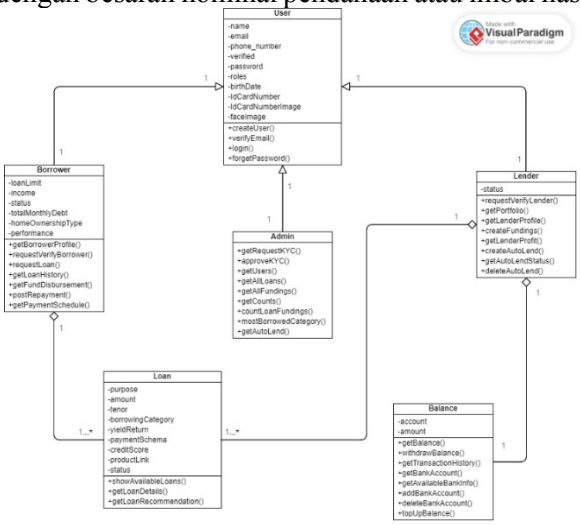


Gambar. 4. *ERD* basis data

C. *Class Diagram*

Class diagram yaitu bagian dari *Unified Modeling Language* (*UML*) yang digunakan untuk merepresentasikan struktur atau rancangan kelas dalam suatu sistem[17]. Perancangan pada sistem yang akan dibangun terdapat beberapa class yang mempresentasikan sistem, antara lain yaitu kelas *User, Borrower, Lender, Work, Loan Application, Funding, Portfolio* dan kelas *Loan History*.

Pada Gambar. 5 kelas *Lender* dan *Borrower* merupakan turunan dari kelas *User* dengan tujuan agar *Lender* atau *Borrower* dapat mewarisi setiap *attribute* ataupun *operation* dari kelas *User*. Pada sistem tersebut, kelas *Borrower* terhubung dengan kelas *Work* yang mana setiap *borrower* dapat memiliki satu pekerjaan yang dapat dicantumkan pada aplikasi. *Borrower* dapat melakukan lebih dari satu peminjaman atau *loans* jika tidak sedang dalam masa peminjaman, sedangkan untuk peminjaman atau *loans* dapat didanai (*Funding*) oleh satu *Lender* dan *Lender* akan mendapatkan imbal hasil sesuai dengan besaran nominal pendanaan atau imbal hasil yang diberikan oleh *Borrower*.



Gambar. 5. Rancangan *class* diagram

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 9, No. 2, Juni 2024, Pp. 568-577

D. Implementasi

Pada implementasi pengembangan *backend API* dengan metode *Test Driven Development* dilakukan untuk menyempurnakan aplikasi yang terhindar dari berbagai *bugs* yang dapat menggangu proses *request* atau *response* dari *API* yang digunakan.

1) Write Test

Dalam *Test Driven Development*, *write test* merupakan salah satu tahap yang akan dilakukan dengan mulai membuat *integration test* sebelum mengkodekan implementasi sebenarnya [18]. Fokus utamanya adalah merancang dan memverifikasi bahwa kode yang akan dikembangkan akan sesuai dengan persyaratan yang telah ditentukan sebelumnya. Tahap awal penulisan *integration test* akan menghasilkan *test* yang gagal atau *error*, hal tersebut dikarenakan belum adanya implementasi yang dibutuhkan pada setiap proses yang dilakukan test.

Pada Gambar. 6 memperlihatkan *integration testing* yang sudah dibuat, didalamnya terdapat beberapa skenario yang berupa pendaftaran akun jika berhasil dan skenario pendaftaran akun ketika gagal, selain itu terdapat skenario untuk melakukan *login* dan verifikasi *email* dengan kasus berhasil dan gagal.

```
FAIL src/tests/integrations/authentication.test.js
  Authentication ~ Positive Case
    × POST /authentication/register should create new user (61 ms)
    × POST /authentication/verification/email/{userId}/{token} should verify email (10 ms)
    × POST /authentication/login should get access token (6 ms)
  Authentication ~ Negatif Case
    × POST /authentication/register should return 422 (9 ms)
    × POST /authentication/register should return 409 for existing email (6 ms)
    × POST /authentication/register should return 422 for invalid password validation (6 ms)
    × POST /authentication/verification/email/{userId}/{token} return 409 already verified (6 ms)
    × POST /authentication/login should return 404 for user not found (9 ms)
    × POST /authentication/login should return 422 invalid payload (7 ms)
    × POST /authentication/login should return 403 invalid password (8 ms)
```

Gambar. 6. *Write test authentications*

```
FAIL src/tests/integrations/borrower.test.js (49.246 s)
  Borrowers ~ Positive Case
    × POST /borrowers/loan should create new loan (2715 ms)
    × GET /borrowers/payment/schedule should return payment schedule data (2715 ms)
    × GET /borrowers/payment/loan should return loan history (2705 ms)
    × PUT /borrowers/request/verification should request verify KYC (2722 ms)
    × GET /borrowers/loan/disbursement should return loan ready to disbursement (2721 ms)
    × POST /borrowers/loan/disbursement should disbursement loan amount to user bank account (2687 ms)
  Borrower ~ Negatif Case
    × POST /borrowers/loan should return 403 not login (2706 ms)
    × POST /borrowers/loan should return 422 invalid payload (2708 ms)
    × POST /borrowers/loan should return 400 invalid minimum imbal hasil (2696 ms)
    × GET /borrowers/payment/schedule should return 403 not log in (2713 ms)
    × GET /borrowers/loan should return 403 not log in (2711 ms)
    × PUT /borrowers/request/verification should return 403 not login (2713 ms)
    × PUT /borrowers/request/verification should return 422 invalid payload (2697 ms)
    × GET /borrowers/loan/disbursement should return 403 not login (2711 ms)
    × POST /borrowers/loan/disbursement should return 403 not login   (2684 ms)
    × POST /borrowers/loan/disbursement should return 404 loanId not found (2718 ms)
    × POST /borrowers/loan/disbursement should return 404 bankId not found (2709 ms)
```

Gambar. 7. *Write test borrower*

```
FAIL src/tests/integrations/lender.test.js
  Lenders ~ Positive Case
    × GET /lenders/profile should return lender data (35 ms)
    × POST /lenders/funding should funding a loan (22 ms)
    × GET /lenders/profit should return lender"s profit (9 ms)
    × GET /lenders/funding should return lender"s portfolio (6 ms)
    × POST /lenders/funding/auto should create auto lend (6 ms)
    × PUT /lenders/request/verification should request verify kyc (5 ms)
  Lenders ~ Negatif Case
    × GET /lenders/profile should return 403 not login (8 ms)
    × POST /lenders/funding should return 403 not login (5 ms)
    × POST /lenders/funding should return 402 balance not enough (7 ms)
    × POST /lenders/funding should return 404 not found (6 ms)
    × POST /lenders/funding should return 404 loan not found (6 ms)
    × POST /lenders/funding should return 422 payload invalid (5 ms)
    × GET /lenders/profit should return 403 not login (5 ms)
    × GET /lenders/funding should return 403 not login (6 ms)
    × POST /lenders/funding/auto should return 403 for not login (5 ms)
    × POST /lenders/funding/auto should return 422 for invalid payload (4 ms)
    × PUT /lenders/request/verification should return 403 not log in (5 ms)
    × PUT /lenders/request/verification should return 422 payload invalid
```

Gambar. 8. *Write test lender*

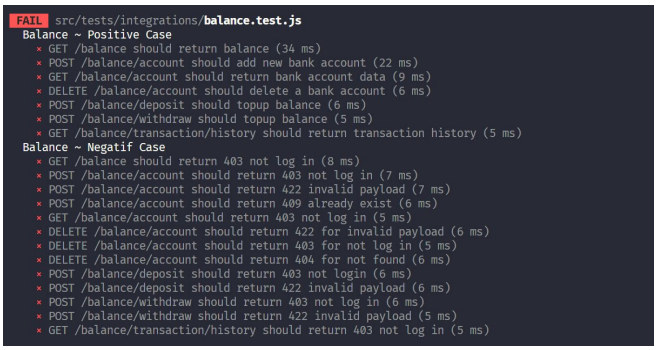
\

```
FAIL src/tests/integrations/loans.test.js
  Loans ~ Positive Case
    × GET /loans/available should return available loans (34 ms)
    × GET /loans/available/{loanId} should return detail loans (8 ms)
    × GET /loans/available/recommended should return recommended loans (5 ms)
  Loans ~ Negatif Case
    × GET /loans/available/{loanId} should return 404 loan not found (8 ms)
    × GET /loans/available/recommended should return 403 not login (5 ms)
```

Gambar. 9. *Write test loans*

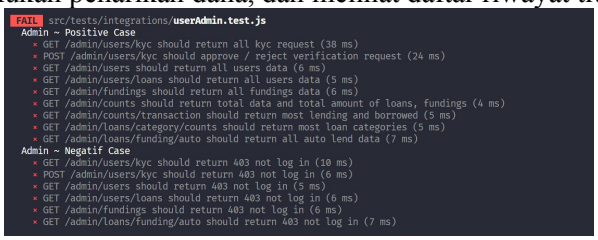
Pada Gambar. 7 dan Gambar. 8 memperlihatkan *integration testing* yang sudah dibuat, skenario yang dapat terjadi pada gambar diatas antara lain yaitu *borrower* dan *lender*. Pada Gambar. 9 memperlihatkan *test* yang sudah dibuat pada layanan *loans*. Layanan ini memiliki beberapa skenario antara lain mendapatkan daftar pinjaman tersedia, mendapatkan detail pinjaman, serta menampilkan rekomendasi pinjaman untuk didanai.

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 9, No. 2, Juni 2024, Pp. 568-577



Gambar. 10. *Write test balance*

Pada Gambar. 10 memperlihatkan hasil dari penulisan *test* pada layanan yang terkait dengan *balance*. Skenario pada layanan ini antara lain mendapatkan saldo, melihat, menambahkan, serta menghapus rekening yang sudah tercantum pada platform, melakukan penarikan dana, dan melihat daftar riwayat transaksi.

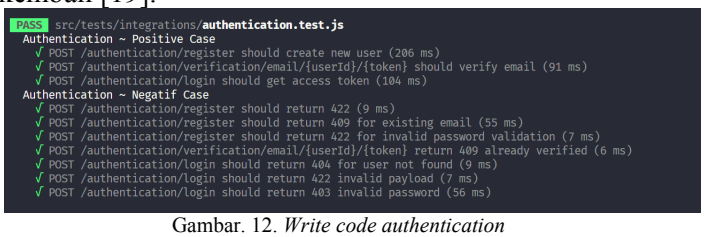


Gambar. 11. *Write test* admin

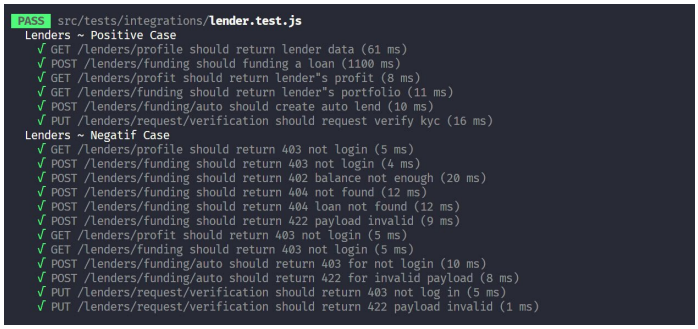
Gambar. 11 menunjukan *test* yang sudah dituliskan pada layanan yang terkait dengan pengguna hak akses admin. Pada layanan ini skenario yang dapat terjadi seputar rangkuman data – data yang terjadi pada platform seperti data pinjaman, pendanaan, dan sebagainya. Selain itu, disini juga terdapat skenario untuk menyetujui ataupun penolakan permintaan verifikasi data *kyc*.

2) Write Code

Setelah tahap *write test* dalam *Test-Driven Development* (*TDD*), langkah selanjutnya yaitu melakukan *write code* atau menuliskan kode implementasi sampai dengan *integration test* yang sudah dibuat pada tahap sebelumnya *pass* atau tidak mengalami *error* kembali [19].

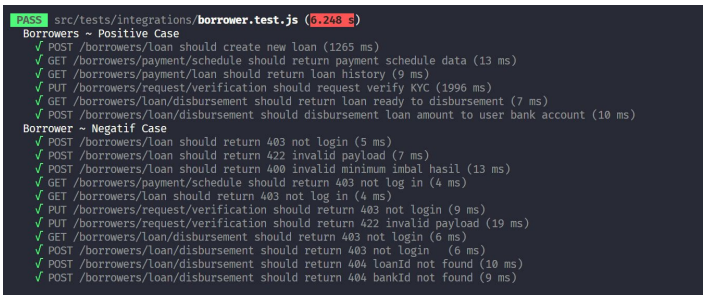


Gambar. 12. *Write code authentication*



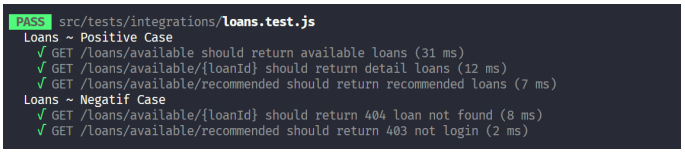
Gambar. 13. *Write code lender*

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 9, No. 2, Juni 2024, Pp. 568-577

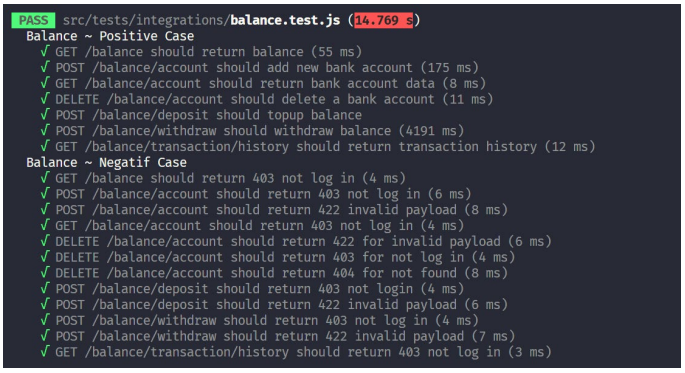


Gambar. 14. *Write code borrower*

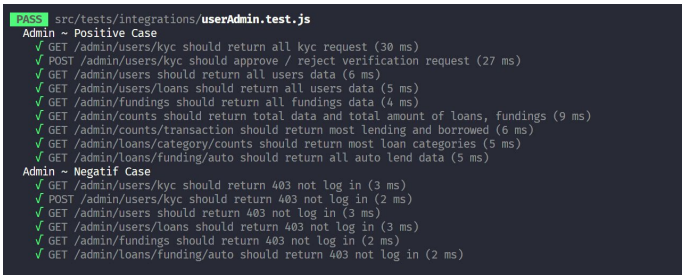
Gambar. 12 menunjukan hasil implementasi yang dilakukan pada *endpoint authentication*, hasil ini merupakan kode program yang sudah memenuhi kriteria pada spesifikasi *test case* yang sudah dibuat, selain itu pada Gambar. 13 menunjukan implementasi untuk *endpoint lender* yang berisi aktivitas – aktivitas yang dapat dilakukan oleh pengguna dengan peran *lender*. Sedangkan, pada Gambar. 14 berisi implementasi kode program untuk *endpoint* pengguna dengan peran *borrower*.



Gambar. 15. *Write code loans*



Gambar. 16. *Write code balance*



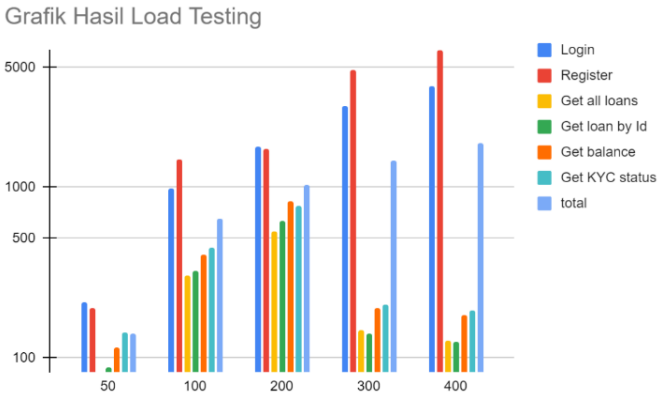
Gambar. 17. *Write code* admin

Pada Gambar. 15 berisi mengenai implementasi yang memenuhi *test* yang ada pada *endpoint loans*. Skenario tes yang ada berupa skenario berhasil dan gagal yang dapat terjadi pada sistem. Pada Gambar. 16 merupakan implementasi pada layanan *balance* yang bertugas untuk mengatur segala aktivitas keuangan dengan akenario berhasil dan gagal. Sedangkan pada Gambar. 17 berisi mengenai skenario tes yang dapat terjadi pada pengguna dengan peran admin.

E. Load Testing

Tahap pengujian dilakukan setelah tahap pengembangan selesai, tahap ini dilakukan dengan tujuan untuk mengetahui segala fungsionalitas yang telah dikerjakan berjalan dengan yang diharapkan. Pengujian dilakukan dengan menggunakan *integration testing* yang telah dilakukan pada saat fase pengembangan pada metode *TDD* dan pada tahap ini akan dilakukan *load testing*. *Load testing* [20] merupakan proses pengujian yang dilakukan pada perangkat lunak atau sistem untuk mengevaluasi seberapa baik dan andalnya performa sistem ketika dihadapkan pada beban tinggi. Tujuan utama dari *load testing* adalah untuk menemukan batasan kemampuan

sistem, mengukur waktu *respons*, serta mengidentifikasi potensi masalah kinerja yang mungkin terjadi saat sistem digunakan dalam situasi yang mendekati atau melebihi beban maksimalnya.



Gambar. 18. Hasil *load testing*

Gambar. 18 memperlihatkan hasil *load testing* yang sudah dilakukan, hasil yang didapatkan berjalan dengan baik dan stabil pada masing – masing *test* yang dilakukan. Pengujian dengan 50 pengguna mendapatkan rata – rata waktu *response* 138 *milisecond*, 100 pengguna dengan rata – rata waktu *response* 650 *milisecond*, serta pada pengujian dengan 200, 300, dan 400 pengguna mendapatkan waktu *response* rata – rata yaitu 1029, 1420, 1803 *milisecond*. Pada pengujian *load test* baru terjadi *error* pada saat melakukan dengan 400 pengguna yaitu pada saat *register total* pengguna mengalami *error* sebanyak 12 pengguna atau 3%. Berdasarkan hasil tersebut aplikasi berjalan dengan baik dan stabil pada jumlah pengguna 50 hingga 300 pengguna.

Berdasarkan hasil pengujian yang dilakukan, pengembangan *API* menggunakan metode *Test Driven Development* menemukan bahwa sistem yang dibangun dapat menampung pengguna yang mengakses atau *hit* suatu *endpoint API* sampai dengan 300 pengguna secara bersamaan tanpa adanya ganggunan yang dihadapi oleh pengguna dengan hanya memerlukan waktu 138 *ms* sampai 1420 *ms* untuk mengakses konten yang dipilih. Selain itu, sistem yang dikembangkan sudah terhindar dari *bugs* atau kecacatan karena berkat adanya *integrations testing* yang dilakukan pada saat implementasi atau pengembangan sistem dengan *Test Driven Development*. Hasil tersebut dapat memenuhi gap yang ada serta tujuan dilakukannya penelitian ini yaitu mengembangkan *API* yang terhindar dari kecacatan dan *response time* yang tidak menggangu pengguna dalam melakukan operasi yang ada pada sistem aplikasi yang dibangun.

IV. KESIMPULAN

Melalui penelitian ini, penggunaan metode *Test Driven Development* (*TDD*) dalam pengembangan perangkat lunak, khususnya dalam membangun *REST API*, terbukti memberikan manfaat yang sangat berarti dalam memastikan kualitas dan performa sistem. Pendekatan *TDD* dengan fokus pada pengujian sejak awal pengembangan membantu mencegah munculnya masalah kompleks yang sulit diperbaiki pada tahap akhir. Selain itu, hasil dari *load testing* juga memberikan keyakinan kuat bahwa sistem ini dapat diandalkan dari segi kecepatan waktu respon dan mampu menangani beban pengguna dengan baik, memastikan pengalaman pengguna yang baik dalam interaksi dengan aplikasi. Penggunaan *Test Driven Development* bukan hanya sekadar metode pengembangan, tetapi juga strategi yang berharga dalam menciptakan perangkat lunak berkualitas tinggi dan handal.

DAFTAR PUSTAKA

[1] T. A. Safitri, "The Development of Fintech in Indonesia," in *Proceedings of the 1st Borobudur International Symposium on Humanities, Economics and Social Sciences (BIS-HESS 2019)*, Paris, France: Atlantis Press, 2020. doi: 10.2991/assehr.k.200529.139.
[2] E. F. Lova, "Financial Technology Peer To Peer Lending Syariah: Sebuah Perbandingan Dan Analisis," *Jeblr*, vol. 1, no. 2, 2021.
[3] M. Pişkin and M. C. Kuş, "Islamic Online P2P Lending Platform," in *Procedia Computer Science*, 2019. doi: 10.1016/j.procs.2019.09.070.
[4] Muh. Arafah, "Peluang Dan Tantangan Pembiayaan Online Syariah Dalam Menghadapi Pinjaman Online Ilegal," *IQTISHADUNA: Jurnal Ilmiah Ekonomi Kita*, vol. 11, no. 1, pp. 65–77, Jun. 2022, doi: 10.46367/iqtishaduna.v11i1.540.
[5] H. M. P. Subardi, "Mekanisme Pembiayaan Fintech Peer to Peer Lending Syariah Bagi UMKM di Indonesia," *Jurnal Produktivitas*, vol. 8, no. 2, 2021, doi: 10.29406/jpr.v8i2.3458.
[6] S. T. Wulandari and K. Nasik, "Menelisik Perbedaan Mekanisme Sistem Peer to Peer Lending pada Fintech Konvensional dan Fintech Syariah di Indonesia," *Nuris Journal of Education and Islamic Studies*, vol. 1, no. 2, 2021, doi: 10.52620/jeis.v1i2.7.
[7] R. A. E. Wahyuni and B. E. Turisno, "PRAKTIK FINANSIAL TEKNOLOGI ILEGAL DALAM BENTUK PINJAMAN ONLINE DITINJAU DARI ETIKA BISNIS," *Jurnal Pembangunan Hukum Indonesia*, vol. 1, no. 3, 2019, doi: 10.14710/jphi.v1i3.379-391.
[8] F. A. Aziz, "Menakar Kesyariahan Fintech Syariah di Indonesia," *Al-Manahij: Jurnal Kajian Hukum Islam*, vol. 14, no. 1, 2020, doi: 10.24090/mnh.v14i1.3567.

[9] N. L. P. P. Dewi and A. A. N. E. S. Gorda, "INTENSI MINAT KAUM MILENIAL DALAM MENGADOPSI LAYANAN PINJAMAN ONLINE (PEER TO PEER LENDING)'," 2022, Accessed: Jan. 17, 2023. [Online]. Available: http://jurnal.stie-aas.ac.id/index.php/jap
[10] M. M. Moe, "Unit Test using Test-Driven Development Approach to Support Reusability," *International Journal of Trend in Scientific Research and Development*, vol. Volume-3, no. Issue-3, pp. 194–196, Apr. 2019, doi: 10.31142/ijtsrd21731.
[11] Mark Masse, *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. IEEE, 2011.
[12] W. Bissi, A. G. Serra Seca Neto, and M. C. F. P. Emer, "The effects of test driven development on internal quality, external quality and productivity: A systematic review," *Inf Softw Technol*, vol. 74, pp. 45–54, Jun. 2016, doi: 10.1016/j.infsof.2016.02.004.
[13] D. Staegemann *et al.*, "A Literature Review on the Challenges of Applying Test-Driven Development in Software Engineering," *Complex Systems Informatics and Modeling Quarterly*, no. 31, pp. 18–28, Jul. 2022, doi: 10.7250/csimq.2022-31.02.
[14] I. B. Kerthyayana Manuaba, "Combination of Test-Driven Development and Behavior-Driven Development for Improving Backend Testing Performance," *Procedia Comput Sci*, vol. 157, pp. 79–86, 2019, doi: 10.1016/j.procs.2019.08.144.
[15] F. Doglio, *REST API Development with Node.js*. Berkeley, CA: Apress, 2018. doi: 10.1007/978-1-4842-3715-1.
[16] F.- Sonata, "Pemanfaatan UML (Unified Modeling Language) Dalam Perancangan Sistem Informasi E-Commerce Jenis Customer-To-Customer," *Jurnal Komunika : Jurnal Komunikasi, Media dan Informatika*, vol. 8, no. 1, p. 22, Jun. 2019, doi: 10.31504/komunika.v8i1.1832.
[17] A. Tazin and M. M. Kokar, "Composition of UML Class Diagrams Using Category Theory and External Constraints," *Journal of Software Engineering and Applications*, vol. 15, no. 12, 2022, doi: 10.4236/jsea.2022.1512025.
[18] A. Roman and M. Mnich, "Test-driven development with mutation testing – an experimental study," *Software Quality Journal*, vol. 29, no. 1, 2021, doi: 10.1007/s11219-020-09534-x.
[19] A. Nanthaamornphong and J. C. Carver, "Test-Driven Development in scientific software: a survey," *Software Quality Journal*, vol. 25, no. 2, 2017, doi: 10.1007/s11219-015-9292-4.
[20] M. Latah and L. Toker, "Load and stress testing for SDN's northbound API," *SN Appl Sci*, vol. 2, no. 1, 2020, doi: 10.1007/s42452-019-1917-y.